
NOT JUST FOR KIDS!
EDWARD H. CARLSON
DATAMOST $19.95
KIDS & THE TI 99/4A
POW
5 FOR Y=1 TO 5
10 FOR X=1 TO 255
20 PRINT CHR$(X);
30 NEXT X
40 NEXT Y
50 END

KIDS AND THE TI 99/4A

BY
Edward H. Carlson

Department of Physics and Astronomy
Michigan State University

Illustrated by
Paul D. Trap

DATAMOST™
8943 Fullbright Avenue
Chatsworth, California 91311

DATAMOST™

ISBN 0-88190-000-1

TABLE OF CONTENTS

INTRODUCTION

GRAPHICS, GAMES, AND ALL THAT

ADVANCED PROGRAMMING

ACKNOWLEDGEMENTS

My sincere thanks go to Paul Sheldon Foote for suggesting I write this book.

I helped prepare and teach in "The Computer Camp" summer camp at Michigan State University for these last two summers. I am deeply grateful to my fellow teachers and board members at the summer camp: Mark Lardie, Mary Winter, and John Forsyth, each of whom shared their teaching experiences with me and suggested techniques for communicating the material in an effective way.

Mark Lardie has been especially generous in reading the typescript and offering suggestions from his extensive experience in teaching computing to children under a variety of formats.

Remembering the enthusiastic pleasure of the summer camp students has encouraged me during the months spent in preparing this book.

Several families have used the first version of this book in their homes and offered suggestions for improvement. I especially wish to thank Steve Peter and his girls Karen and Kristy; George Campbell and his youngsters Andrew and Sarah; Beth O'Malia and Scott, John and Matt; Chris Clark and Chris Jr., Tryn, Daniel, and Vicky; and Paul Foote and David.

John Decarli of the Computer Mart in East Lansing has helped me cope with the inevitable equipment breakdown crises.

My children offered ever present examples of juvenile taste and tested lessons up to the tantrum limit. With my wife, they recognized my need for long periods in the writing den and for quiet in the house. So my final and heartfelt thanks go to my wife Louise and our children Karen, Brian and Minda.

TO THE KIDS

This book teaches you how to write programs for the TI 99/4A computer.

You will learn how to make your own action games, board games and word games. You may entertain your friends with challenging games and provide some silly moments at your parties with short games you invent.

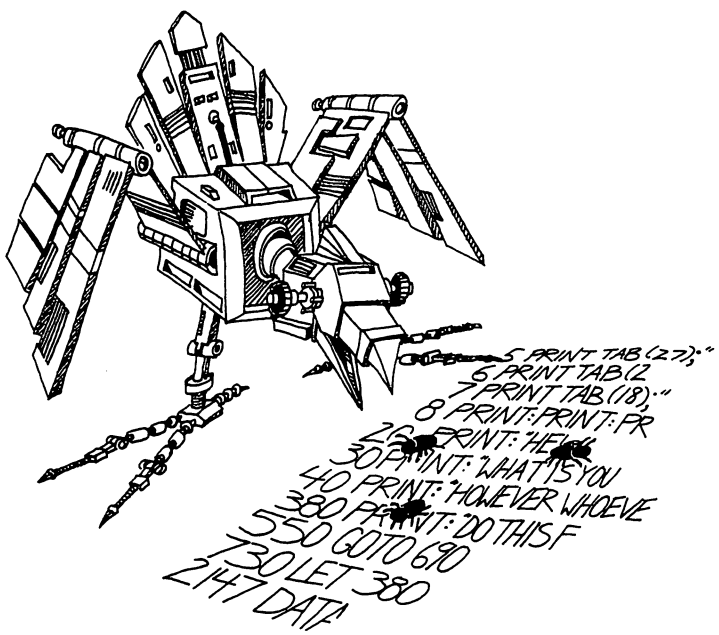
Perhaps your record collection or your paper route needs the organization your special programs can provide. If you are working on the school yearbook, maybe a program to handle the finances or records would be useful.

You may help your younger sisters and brothers by writing drill programs for arithmetic facts or spelling. Even your own schoolwork in history or foreign language may be made easier by programs you write.

How to Use This Book: Do all the examples. Try all the assignments. If you get stuck, first go back and reread the lesson carefully from the top. You may have overlooked some detail. After trying hard to get unstuck by yourself, you may go ask a parent or teacher for help.

There are review questions for each lesson. Be sure you can answer them before announcing that you have finished the lesson!

MAY THE BLUEBIRD OF HAPPINESS EAT ALL THE BUGS IN YOUR PROGRAMS!



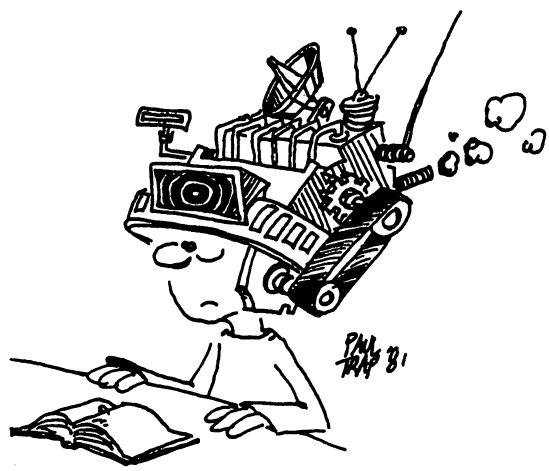
TO THE PARENTS

This book is designed to teach TI BASIC to youngsters in the range from 10 to 14 years old. It gives guidance, explanations, exercises, reviews and "quizzes." Some exercises have room for the student to write in answers that you can check later. Answers are provided in the back of the book for assignments. Your child will probably need some help in getting started and a great deal of encouragement at the sticky places.

Learning to program is not easy because it requires handling some sophisticated concepts. It also requires accuracy and attention to detail which are not typical childhood traits. For these very reasons it is a valuable experience for children. They will be well rewarded if they can stick with the book long enough to reach the fun projects which are possible once a repertoire of commands is built up.

How to Use The Book: The book is divided into 33 lessons for the kids to do. Each lesson is preceded by a NOTES section which you should read. It outlines the things to be studied, gives some helpful hints, and provides questions which you can use verbally (usually at the computer) to see if the skills and concepts have been mastered.

These notes are intended for the parents, but the older students may also profit by reading them. The younger students will probably not read them, and can get all the material they need from the lessons themselves. For the youngest children, it may be advisable to read the lesson out loud with them and discuss it before they start work.



TO THE TEACHER

This book is designed for students in about the 7th grade. It teaches TI BASIC on cassette systems. The lessons contain explanations (including cartoons), examples, exercises and review questions. Notes for the instructor which accompany each lesson summarize the material, provide helpful hints and give good review questions.

The book is intended for self study, but may also be used in a classroom setting.

I view this book as teaching programming in the broadest sense, using the BASIC language, rather than teaching "BASIC." Seymour Papert has pointed out in MINDSTORMS that programming can teach powerful ideas. Among these is the idea that procedures are entities in themselves. They can be named, broken down into elementary parts, and debugged. Some other concepts include these: "chunking" ideas into "mind sized bites," organizing such modules in a hierarchical system, looping to repeat modules, and conditional testing (the IF...THEN statement).

Each concept is tied to everyday experiences of the student through choice of language to express the idea, through choice of examples and through cartoons. Thus metaphor is utilized in making the "new" material familiar to the student.

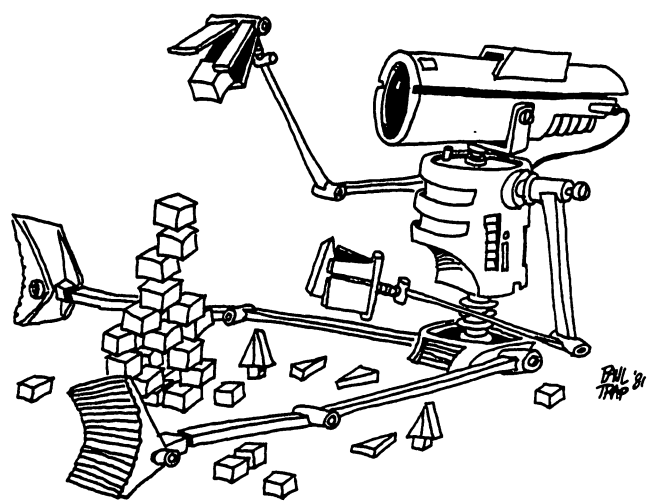
ABOUT PROGRAMMING

There is a common misconception that programming a computer must be very similar to doing arithmetic. Not so. The childhood activities that computing most resembles may be playing with building blocks and writing an English composition.

Like a set of blocks, BASIC uses many copies of a small number of elements (commands) which are combined in rather standardized ways to achieve an original end result. As familiarity with the system grows, a "bag of tricks" is collected by the programmer which allows each command to serve a larger number of functions, just as the child first uses the "triangle block" in making roofs but later finds that two of them make a splendid fir tree.

Like an essay, a program is a finished product which fulfills a specified need. The child writing to the theme "How I spent my summer," adopts one of several working styles. The beginner may be hung up in how to hold the pencil and how to spell. The same child a few grades later will just start writing, not spending much time in forming good paragraphs, much less in planning the overall structure of the composition. With maturity comes freedom to move back and forth among the levels of concern, now thinking about the overall form, and a few moments later paying attention to word choice or punctuation.

Computing does have some similarities to arithmetic as seen by most children. There are rigid rules to learn: procedures in arithmetic but only syntax in programming. Even the tiniest mistake makes the whole result "wrong." (A more effective attitude in programming is that "wrong" results are partly right and need debugging, a normal and expected activity.) However, the limited scope for creativity in arithmetic contrasts sharply with the emphasis on creativity in program writing.



Programming offers general educational advantages not easily found elsewhere in a child's experiences. The plasticity of the form, words on a screen which are created and destroyed by the touch of a key, allows effort to be concentrated on the central features to be learned. These features are balanced between analysis (why doesn't it work as I want) and synthesis (planning on several size scales, from the program as a whole down through loops and subroutines to individual commands). Learning on the computer is efficient of effort. Errors of syntax are automatically pointed out by the computer.

The analytical and synthetical skills learned in programming can be transferred to more general situations and can help the child to a more mature style of thinking and working.

ABOUT THE BOOK

The book is arranged in 33 lessons, each with notes to the instructor and each containing assignments and review questions.

For instructors who feel themselves weak in BASIC or are beginners, the student's lessons form a good introduction to BASIC. The lessons and notes differ in style. The lessons are pragmatic and holistic, the notes and GLOSSARY are detailed and explanatory.

The book starts with a bare bones introduction to programming, leading quickly to the point where interesting programs can be written. See the notes for Lesson 5, THE INPUT COMMAND, for an explanation. The central part of the book emphasizes more advanced and powerful commands. The final part of the book continues this, but also deals with broader aspects of the art of programming such as editing and debugging, and user friendly programming.

The assignments involve writing programs, usually short ones. Of course, many different programs are satisfactory "solutions" to these assignments. In the back of the book I have included solutions for assignment problems, some of them written by children who have used the book.

Lessons 14: SAVING TO TAPE and 18: COMMAND AND RUN MODES can be studied anytime after the first lesson.

INTRODUCTION

INSTRUCTOR NOTES 1 NEW, PRINT, REM, AND RUN

This lesson is an introduction to the computer.

The contents of the lesson:

1. Turning on the computer.
2. Typing versus entering commands or lines. ENTER key.
3. The computer understands only a limited number of key words.
4. In this lesson, NEW, PRINT, REM, RUN.
5. What a program is. Numbered lines.
6. NEW clears the memory and the screen.
7. What is seen on the screen and what is in memory are different. This may be a hard concept for the student to grasp at first.
8. RUN makes the computer go to memory, look at the commands in the lines (in order) and perform the commands.
9. One can skip numbers in choosing line numbers, and why one may want to do so.

Key words may be commands, statements, or functions. Most can be either a statement or a command, as the user wishes. I tend to call the latter "commands." Later, after the student has seen many examples of each, the classification of keywords is clarified.

QUESTIONS:

1. Write a program that will print your name.
2. RUN it.
3. Erase the program from memory (and the screen).
4. Try to RUN it. What does the computer say? Why?
5. Why do you usually skip numbers when numbering lines?

LESSON 1 NEW, PRINT, REM, AND RUN

GETTING STARTED

Turn on your computer. You will see:

```
READY--PRESS ANY KEY TO BEGIN
```

Press a key. You will see a menu that says:

```
PRESS

1 FOR T1 BASIC
```

Press key "1." You will see:

```
T1 BASIC READY

>
```

and a flashing square on a blue screen. This square is called the "cursor." When you see it on the screen, the computer wants you to type something.

"Cursor" means "runner." The square runs along the screen showing where the next letter you type will appear.

TYPING

Type some things. What you type shows on the TV screen.

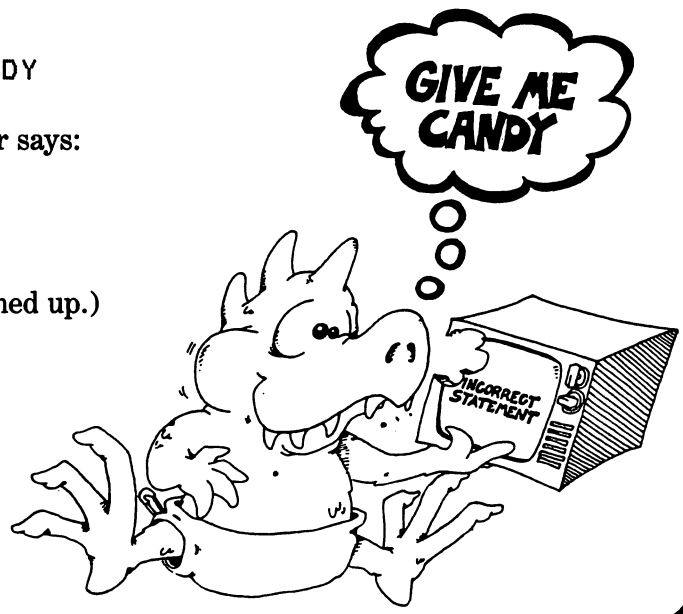
COMMAND THE COMPUTER

Try this. Type: `GIVE ME CANDY`

and press the ENTER key. The computer says:

```
* INCORRECT STATEMENT
```

and sounds a tone from the TV.
(Be sure that you have the TV sound turned up.)



The computer understands only about 80 words. The words are called "key words." You need to learn which words the computer understands.

Here are the first commands to learn:

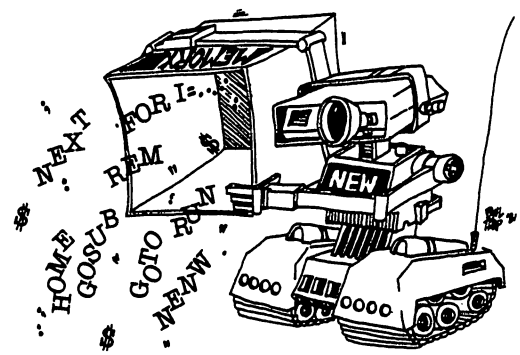
```
NEW    PRINT    REM    RUN
```

THE NEW COMMAND

Type: `NEW`

and press ENTER.

NEW empties the computer's memory so you can put your program in it. It also erases the TV screen.



HOW TO ENTER A LINE

When we say "enter" we will always mean to do these two things:

1) type a line
2) then press the ENTER key.

Enter this line: `10 PRINT "HI"`

The " marks are quotation marks.

To make quotation marks:

hold down the FCTN key and

press the key that has the P and the " on it.

(Did you remember to press the ENTER key at the end of the line?)

Now line number 10 is in the computer's memory.

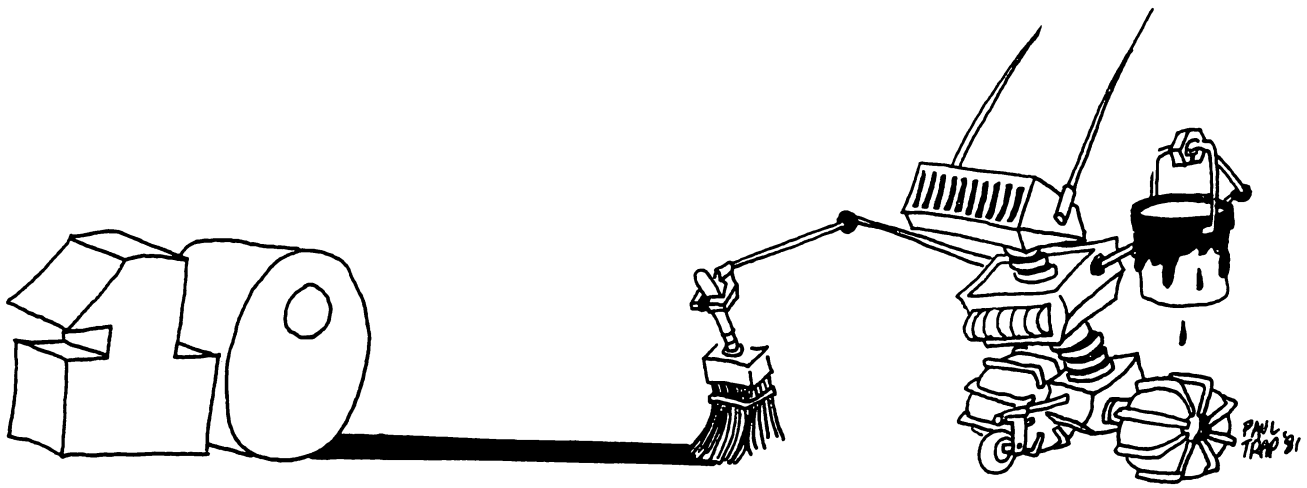
It will stay in memory until:

you enter the NEW command

or you turn off the computer.

or you press FCTN QUIT (more in a later lesson)

Line 10 is a very short program



THE NUMBER ZERO AND THE LETTER "O"

It is easy to get zero and the letter "O" mixed up.

The computer always writes on the screen like this:

the zero is like this:	zero	⌀
and the letter O like this:	letter O	O

This book writes zero like this: zero Ø

Be careful to type zero, not "O," for numbers:

right	`10 PRINT "HI"`
wrong	`10 PRINT "HI"`

WHAT IS A PROGRAM?

A program is a list of commands you wish the computer to do.

The commands are written in lines.

Each line starts with a number.

The program you entered above has only one line.

HOW TO RUN A PROGRAM

A moment ago you put this program in memory:

```
10 PRINT "HI"
```

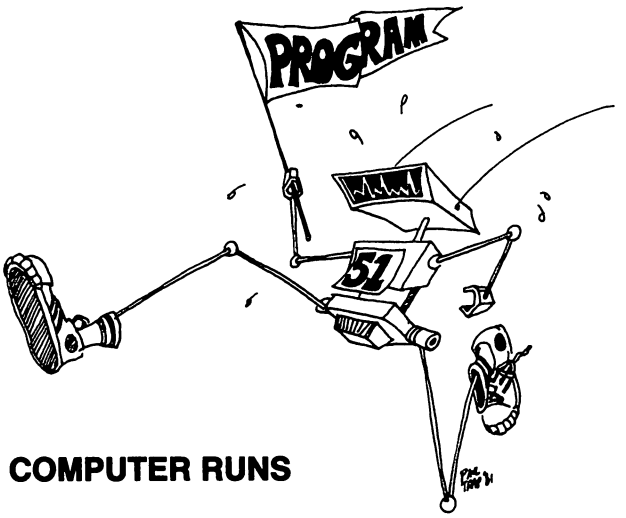
Now enter: RUN

(Did you remember to press the ENTER key?)

The RUN command tells the computer to look into its memory for a program and then to obey the commands it reads in the lines.

Did the computer obey the PRINT command? The PRINT command tells the computer to print whatever is between the quotation marks.

The computer printed: HI



EXTRA STUFF WHILE THE COMPUTER RUNS

The screen turns green while the program is running.

After the program is done, the computer prints

```
** DONE **
```

and then the screen turns blue again.

A LONGER PROGRAM

Clear the memory with `NEW`

(Did you remember to press ENTER afterward?)

Enter this program:

```
1 REM PROGRAM 2
2 PRINT "HI"
3 PRINT "FRIEND"
```

This program has 3 lines. Each line starts with a command.

Enter: `RUN`

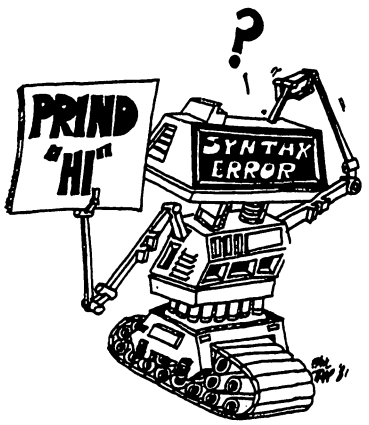
What the program does:

Line 1 The computer skips this line because it is a REM.
Line 2 The computer prints "HI"
Line 3 The computer prints "FRIEND"

The REM command lets you put little notes to yourself in the program. REM means "remark" or "reminder."

In line 1 we used REM to give a name to the program. The name is "PROGRAM 2."

The computer does the commands in the lines. It starts with the lowest line number and goes down the list in order.



HOW TO NUMBER THE LINES IN A PROGRAM

Usually you will skip numbers when writing the program.

Like this:

```
10 REM PROGRAM 2
20 PRINT "HI"
30 PRINT "FRIEND"
```

It is the same program but has different numbers.

The numbers are in order, but some numbers are skipped.

You skip numbers so that you can put new lines in between the old lines later if the program needs fixing.

Assignment 1:

1. Use the command NEW. Explain what it does.
2. Write a program that uses REM once and PRINT twice. Then use the command RUN to make the program obey the commands.
3. What is the difference between "entering" a line and "typing" something?

INSTRUCTOR NOTES 2 COLOR AND SOUND

This lesson introduces the CALL SCREEN and CALL SOUND statements. We wish to make plenty of "bells and whistles" available to the student to increase program richness.

The idea of a "string constant," used in Lesson 1, is explained. The numbers appearing in a string, for example the "19," cannot be used directly in arithmetic.

The CALL command calls a number of built-in subroutines:

`CALL CLEAR`	erases the screen
`CALL SCREEN`	colors the screen
`CALL SOUND`	generates sounds
`CALL HCHAR`	puts characters on screen
`CALL VCHAR`	same, in vertical lines
`CALL CHAR`	makes hi-res characters
`CALL KEY`	gets char. from keyboard
`CALL JOYST`	reads joystick position
`CALL GCHAR`	reads char. from screen

The CALL subroutine commands are explained one by one as you go through the book.

The CALL SOUND command has 3 arguments.

`CALL SOUND`	(length, pitch, loudness)
length	1 to 4250 is the duration of the sound in milliseconds
pitch	110 to 44732 is the pitch in hertz (cycles per second)
loudness	0 (loud) to 30 (off)

The CALL SOUND command is further explained in a later session.

QUESTIONS:

1. How do you do each of these things:
   Make the computer "peep"?
   Empty the memory and erase the screen?
   Print your name?
2. What is a "string"?
3. What special key do you press to "enter" a line?
4. What is a command? Give some examples.
5. How do you make the screen change color?
6. Write a program to print "FIRE", make the screen turn red, and make the computer peep.

LESSON 2 COLOR AND SOUND

Enter:

```
NEW
```

NEW empties the memory and erases the screen. You are ready to start this lesson.

COLOR THE SCREEN

Enter:

```
10 REM COLOR THE SCREEN
20 CALL SCREEN(14)
```

RUN the program.

Line 20 tells the computer to change the screen color.

Rule: The number in ( ) after SCREEN tells what color the screen will be. Any number from 1 to 16 is OK.

RUN it again trying different numbers.

The color changes back to blue as soon as the program is over.

THE COMPUTER PEEPS LIKE A BIRD

Add this line:

```
30 CALL SOUND(200,1500,10)
```

RUN it.

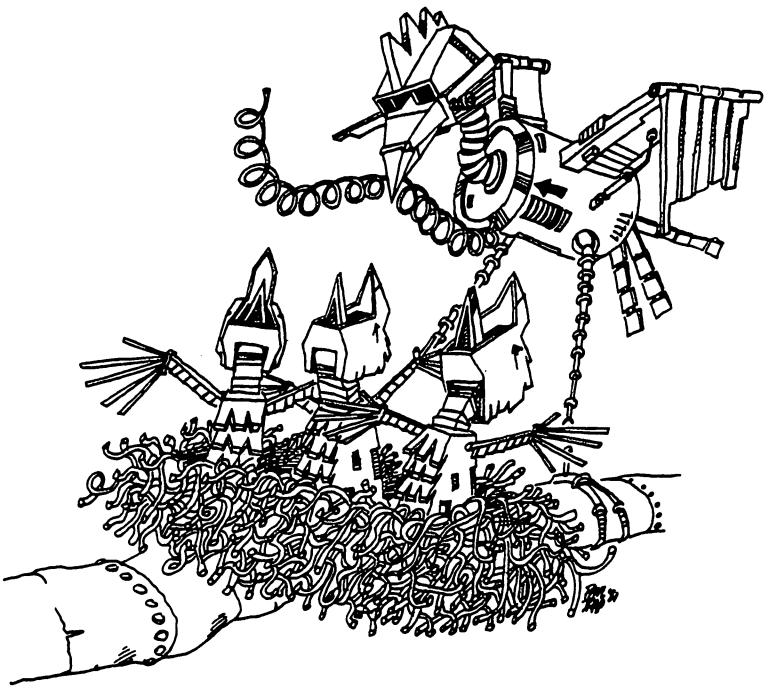
Did you hear it "peep"?

If you do not hear a tone, turn up the sound on your TV.

Put 500 in place of the 200. Now the peep lasts longer.

Put 800 in place of 1500. Now the sound is lower.

Try other numbers.



PRINTING AN EMPTY LINE

Run this:

```
10 REM SOME LINES
20 PRINT"HERE IS A LINE"
30 PRINT
40 PRINT"ONE LINE WAS SKIPPED"
```

Line 30 just prints a blank line.

STRING CONSTANTS

Look at these PRINT statements:

```
10 PRINT "JOE"
10 PRINT "#D47%%*%"
10 PRINT "19"
10 PRINT "3.14159265"
10 PRINT "I'M 14"
10 PRINT " "
```

Letters, numbers and punctuation marks are called "characters." Even a blank space is a character. Look at this:

```
10 PRINT " "
```

Characters in a row make a "string."

The letters are stretched out like beads on a string.

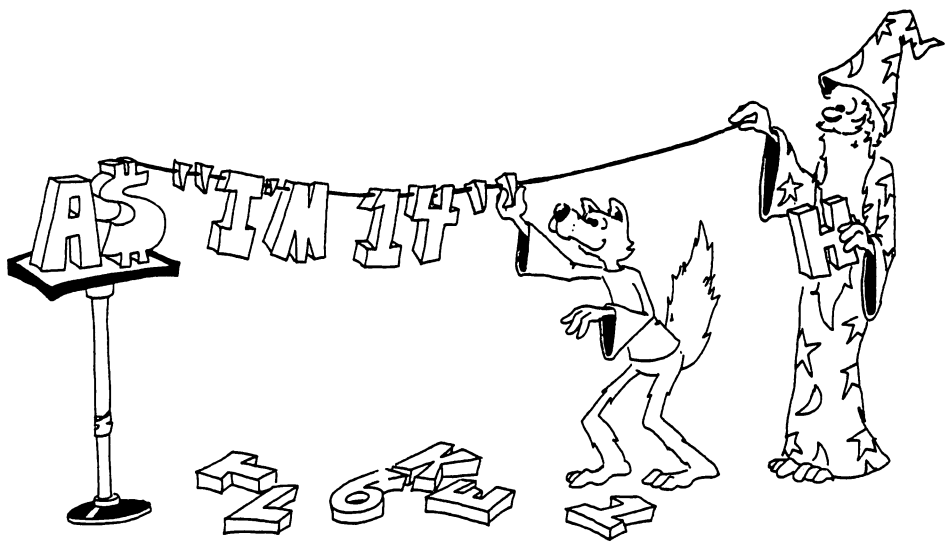
A string between quotation marks is called a "string constant."

It is a string because it is made of letters, numbers and punctuation marks in a row.

It is a constant because it stays the same. It doesn't change as the program runs.

Assignment 2:

1. Write a program that prints your first, middle and last names.
2. Now add a "peep" before it prints each name. Make each "peep" a different tone, deep for your first name, high for your last name.
3. Now make the screen change color for each name.



INSTRUCTOR NOTES 3 CALL CLEAR, LIST

In this lesson:

LIST, LIST 30
REM for titles, remarks
memory boxes holding lines
erase one line from memory
add a line between old lines
replace a line
drawings using PRINT commands

Your student needs to understand that the program is stored in memory even when it is not visible on the screen, and that LIST just lists the program to the screen. The special uses like LIST 100-300 and LIST -300 will be taken up later.

The memory as a shelf of boxes is a key model of the computer that we will develop in this book. It is an important tool in helping the student understand variables and the detailed workings of complicated expressions in a statement.

REM as a remark command can be a little confusing to new students. It needs to be distinguished both from PRINT and from just typing to the screen. Using print to draw pictures is demonstrated. It is better to draw some at the end of each lesson than to do a lot now. Drawing after lesson 4 helps develop line editing skills.

QUESTIONS:

1. How do you erase a line you no longer want?
2. Type CALL CLEAR. Now how do you show all of the program in memory on the screen?
3. How can you replace a wrong line with a corrected one?
4. Suppose you want to put a line in between two lines you already have in memory. How do you do this?
5. Explain how the computer puts program lines into "boxes" in memory. What does it write on the front of the box?

LESSON 3 CALL CLEAR, LIST

Enter: `NEW`

Start each lesson with NEW to erase the memory and the screen.

PUT A PROGRAM INTO MEMORY

Now enter:

```
10 REM HOUSE
20 PRINT "LISTEN"
30 CALL SOUND(200,800,10)
40 CALL SOUND(300,600,10)
50 PRINT "DID YOU HEAR THE DOORBELL?"
```

Run this 5 line program.

ERASING THE SCREEN

Now enter: `CALL CLEAR`

CALL CLEAR is a command to erase the screen. It doesn't erase the program in memory. CLEAR means the same as "erase."

IS THE PROGRAM LOST FOREVER?

You can no longer see the program on the screen. But the program is not lost. The computer had stored the program in its memory. We can ask the computer to show us the program again.

LISTING THE PROGRAM

Enter: `LIST`

and the computer will list the whole program on the screen. To see just one line of the program, ask for it by number:

```
LIST 30
```

shows line 30 of the program.

THE MEMORY

The computer's memory is like a shelf of boxes.

The name of the box goes on the front of each box.

At the start, all the boxes are empty and no box has a name.

When you entered: `10 REM HOUSE`

the computer took the first empty box and wrote the name "Line 10" on the label.

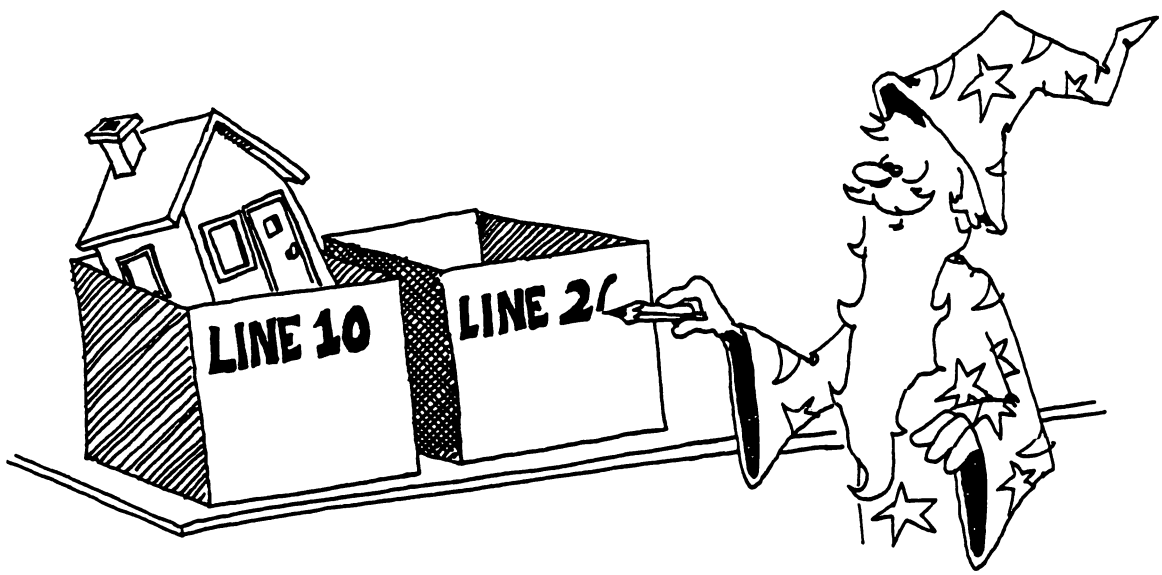
Then it put the command REM HOUSE into the box and put the box back on the shelf.

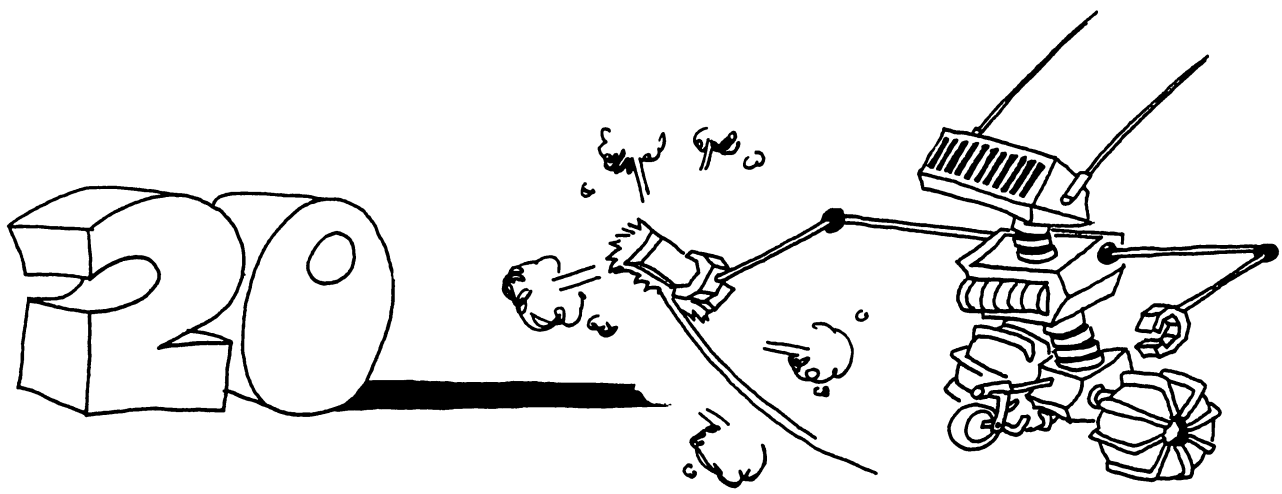
When you entered: `20 PRINT "LISTEN"`

the computer took the second box and wrote "Line 20" on its label.

Then it put the statement PRINT "LISTEN" into the box and put that box into its place on the shelf.

What did the computer do when you entered line 30?





ERASING A LINE FROM MEMORY

To erase one line of the program, enter the line number with nothing after it.

To erase line 20, enter:

```
20
```

You still see the line on the screen.

But do a LIST. You see that line 20 is gone from memory.

When you enter just a line number with nothing after it, the computer

finds the box with that line number on it

empties the box

and erases the name off the front of the box.

What does the computer do to the boxes when you give it the command NEW?

ADDING A LINE

You can add a new line anywhere in the program, even between two old lines. Just pick a line number between any two existing line numbers and type your line in. The computer puts it into the correct place.

Enter NEW and this:

```
10 REM MORE AND MORE
12 PRINT
40 PRINT "MORE LINES WANTED"
```

List it and run it. Now add this line:

```
15 PRINT "STILL"
```

List and run it again.

FIXING A LINE

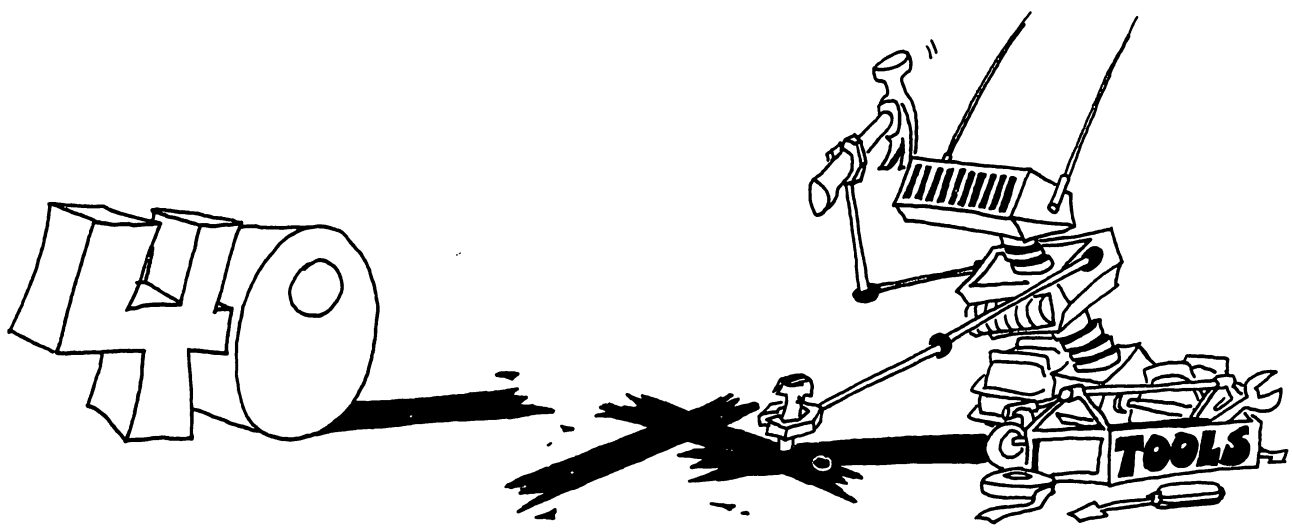
If a line is wrong, just type it over again.

For example, to change line number 40 in the above program:

Enter:

```
40 PRINT "NEEDS FIXING"
```

What did the computer do to the box named "Line 40" when you entered the line?



THE REM COMMAND

Use a REM command to put a title on your program.

Enter:

```
NEW

10 REM REMARKS
20 CALL CLEAR
30 PRINT "LINE 10 DOES NOTHING"
35 REM THIS LINE DOES NOTHING
RUN
```

What happens in each line of the program?

Line 10 .....

Line 20 .....

Line 30 .....

Line 35 .....

REM means "remark," or "reminder."

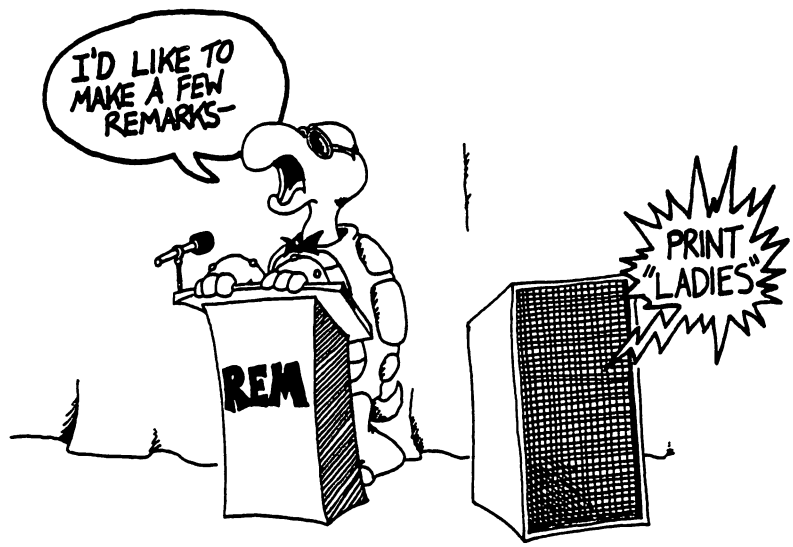
Use REM to give a title to the program

Use REM to write little notes in the program:

the notes are for you when you read the program

the notes are also for other readers.

Make the notes explain how the program works.



PICTURE DRAWING

You can use the PRINT command to draw pictures.

Here is a picture of a car. Enter NEW then enter this program.

```
10 REM STANLEY STEAMER
15 CALL CLEAR
20 PRINT
30 PRINT" XXXXXX"
40 PRINT"XXXXXXXXXXXX"
50 PRINT" O        O"
```

Don't forget to put the spaces in the PRINT lines! They are part of the drawing. Run the program.

Assignment 3:

1. Add a line to the STANLEY STEAMER program to make the car honk its horn.
2. What command will list line 10 of the program?
3. How do you tell the computer to list the whole program on the screen?
4. What does the computer do (if anything) when it sees the REM command?
5. What is the REM command used for?
6. Use CALL CLEAR, CALL SOUND, REM, and PRINT to draw 3 flying birds on the screen. Make each bird peep.

INSTRUCTOR NOTES 4 SPECIAL KEYS

This lesson concerns the arrow keys, the FCTN key, and the DEL key.

The strip overlay belongs in the slot over the "number" row of keys at the top of the keyboard. It should be left there while the student is studying from this book.

The arrow keys are used with FCTN in moving the cursor around in the line currently being worked on. Characters in the line are not affected by the cursor moving over them. Wherever the cursor stops, you can type in new characters, replacing the old ones. When all is satisfactory, the line can be entered in the computer by pressing ENTER.

The deletion and insertion of characters in the current line are a little more tricky. The DEL and FCTN keys are used for DELETE, and are explained in this lesson. INS for INSERT is explained in the next lesson.

These methods work for the line you are currently typing. After you press ENTER, the line is in memory. If you want to change a line already in memory, you must call it up with the EDIT command. This is described in a later lesson.

QUESTIONS:

1. What is a "cursor"? What is it good for?
2. Have your student demonstrate how to edit a line. This includes using the arrow keys to move the cursor to the interior of the line, modifying characters there, and pressing the ENTER key.
3. Have your student demonstrate using the DEL key to delete a character from the middle of a line.
4. Have your student demonstrate making a character repeat on the screen.

LESSON 4 SPECIAL KEYS

THE FCTN KEY

Find the FCTN key. "FCTN" means "function."

The FCTN key is a "helper" key. It helps other keys:

to fix errors in your typing
to stop a program that is running.

THE ARROW KEYS

Find these keys: FCTN

left arrow (on the S key)

right arrow (on the D key)

Now hold down the FCTN key and press the right arrow key.

The cursor moves right one space.

Now hold down both keys:

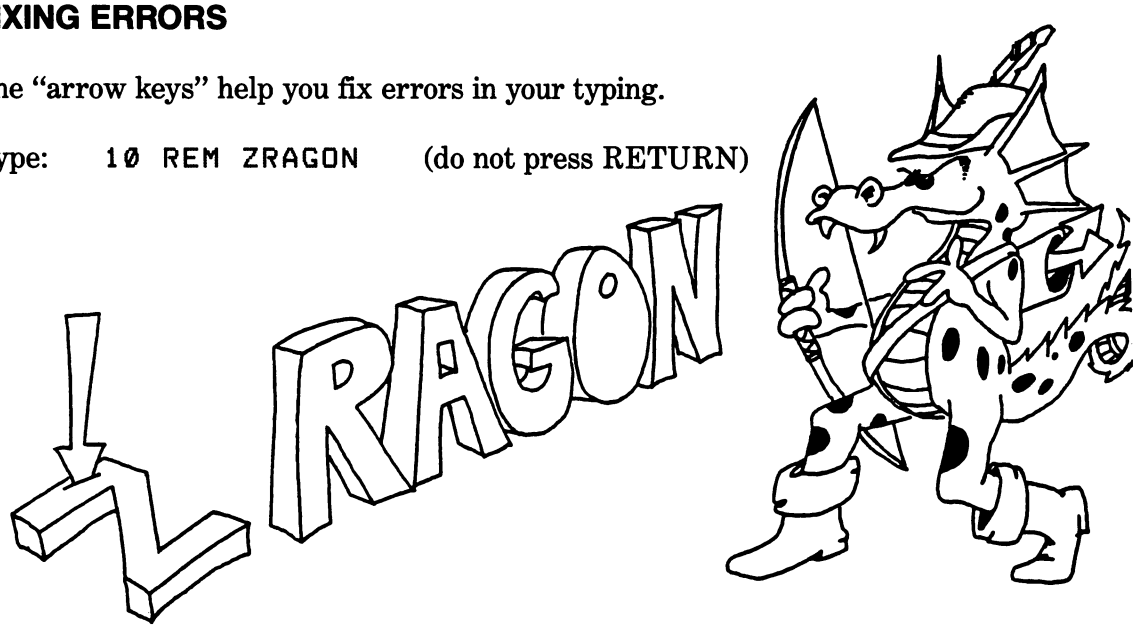
The cursor runs to the right.

Do the same with FCTN and the left arrow key.

FIXING ERRORS

The "arrow keys" help you fix errors in your typing.

Type: `10 REM ZRAGON` (do not press RETURN)



Hold down the FCTN and left arrow keys to move the cursor (the flashing square) over the "Z".

Type a "D" instead.

Now the line is correct, reading:

```
10 REM DRAGON
```

Press ENTER to store the correct line in the memory.

REPEATING A KEY

Hold down the H key. You get a row like this:

```
HHHHHHHHHHHHHHHHHHHHHHHHH
```

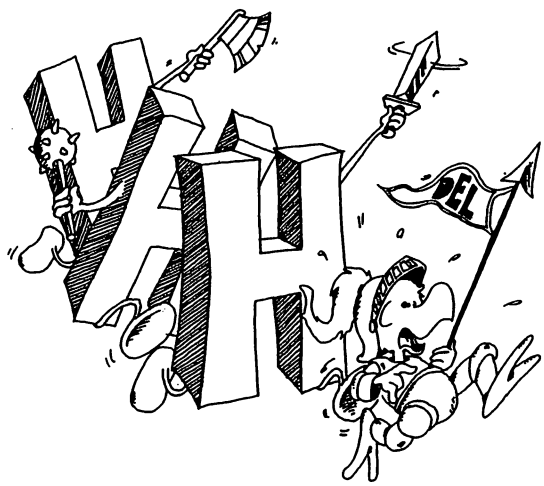
This works for most keys.

Try holding down: the space bar

the ENTER key

the period key.

Repeating a key is useful when drawing pictures.



Assignment 4A:

1. Type a line with your name in it. Misspell your name on purpose. Then use FCTN and the arrow keys to fix the line. When you are done, press ENTER to enter the line.
2. Draw a "smiley face." Use the FCTN and arrow keys. Hold down a key to make a line.

THE STRIP OVERLAY

A special strip of plastic came with your computer.

The strip has words printed along it.

DEL INS ERASE CLEAR BEGIN PROC'D AID REDO BACK QUIT

It belongs in the slot just above the number keys. Put it there now.

The words help to remind you of some special things the keys do.

What key is below DEL on the strip? .....

There is a red dot on the strip.

What key has a red dot? .....

There is a grey dot on the strip.

What key has a grey dot? .....

What key has a yellow dot? .....

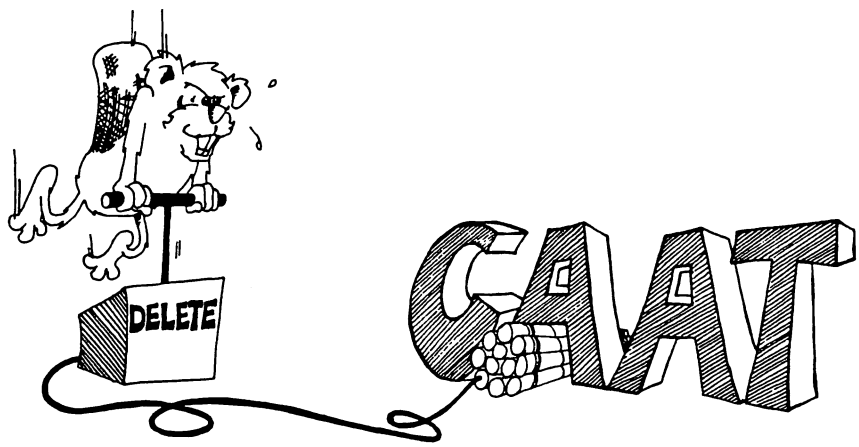
In this book we will study three of these words: DEL, INS, and CLEAR.

THE DELETE KEY

The DEL key is used with the FCTN key to erase letters.

Type this: `10 REM CAAT` (do not press ENTER yet)

Too many "A's" in the CAT.



Move the cursor over the first "A".

Hold down FCTN and press DEL. (DEL is the "1" key.) The "A" disappears.

```
10 REM CAT
```

Press ENTER to enter the line in the program memory.

What happens if you hold down the FCTN and DEL keys?

Assignment 4B:

1. Practice using the FCTN, arrow keys, and DEL key. Type this:

   ```
   10 REM WIZZARD
   ```
   (fix it to "WIZARD")

   Type and fix this:

   ```
   10 REM TTIIGGEERR
   ```
   (make it "TIGER")

INSTRUCTOR NOTES 5 THE INPUT STATEMENT, STRING VARIABLES

This lesson concerns the INPUT statement and the idea of a string variable.

We teach INPUT in its pure form in this lesson: no message in front. In a later lesson we will return to the INPUT statement.

We are quickly outlining the essentials of programming so that the student "sees the forest" and is able to write meaningful programs. The statements required for interesting programs are:

PRINT	allows	output
INPUT		input
GOTO		infinite looping
FOR...NEXT		finite looping, time delays
IF		branching and decisions
RND		random numbers for games

On the second pass, we will elaborate on the basic statements and teach further statements.

Back to this lesson. String variables are introduced using the "box" concept again. Variable names are restricted to one letter for the time being. This avoids confusion in short programs and allows faster typing.

We will work with strings and ignore numbers for as long as possible because strings make for more interesting programs and offer a less confusing entry into the logical concepts of programming.

QUESTIONS:

1. What two different things does the computer put into boxes?
2. How does the program ask a person to type in something?
3. How do you know the computer is waiting for an answer?
4. A letter with a dollar sign after it is called what?
5. Write a short program which uses CALL CLEAR, PRINT and INPUT.
6. What happens if the answer you give to an INPUT has a comma in it?
7. How do you use the DEL key?

LESSON 5 THE INPUT STATEMENT, STRING VARIABLES

Use INPUT to make the computer ask for something.

Enter:

```
10 REM :::: TALKY-TALK ::::
15 CALL CLEAR
20 PRINT "SAY SOMETHING"
25 INPUT A$
30 PRINT
35 PRINT "DID YOU SAY "
40 PRINT A$
```

Run it. When you see a question mark, type "HI" and press the ENTER key.

The question mark was written by INPUT in line 25. The flashing cursor means the computer is waiting for you to type something in.

When you type "HI", the computer stores this word in a box named A$.

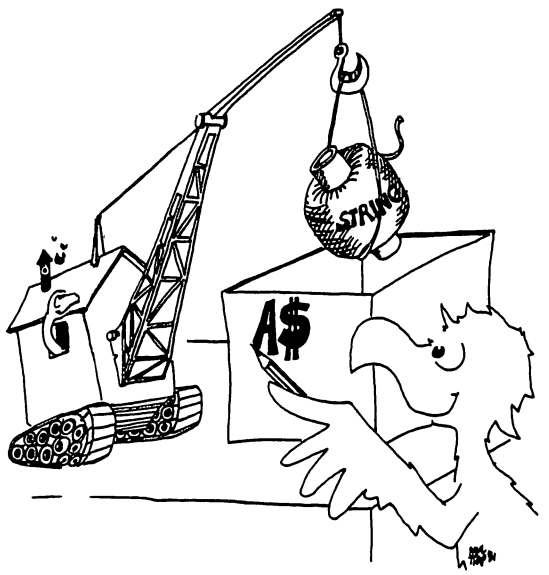
Later, in line 40, the program asks the computer to print whatever is in the box named A$.

Run the program again and this time say something funny.

STRING VARIABLES

A$ is the name of a "string variable." The name is written on the front of a box and the string is put inside the box.

Rule: A string variable name always ends in a dollar sign,"$". You can use any letter you like for the name and then put a dollar sign after it.



A$ is called a variable because you can put different strings into the box at different times in the program. The box can hold only one string at a time.

ERROR MESSAGES FROM INPUT

Run the program again and answer the question with:

```
HI, THERE
```

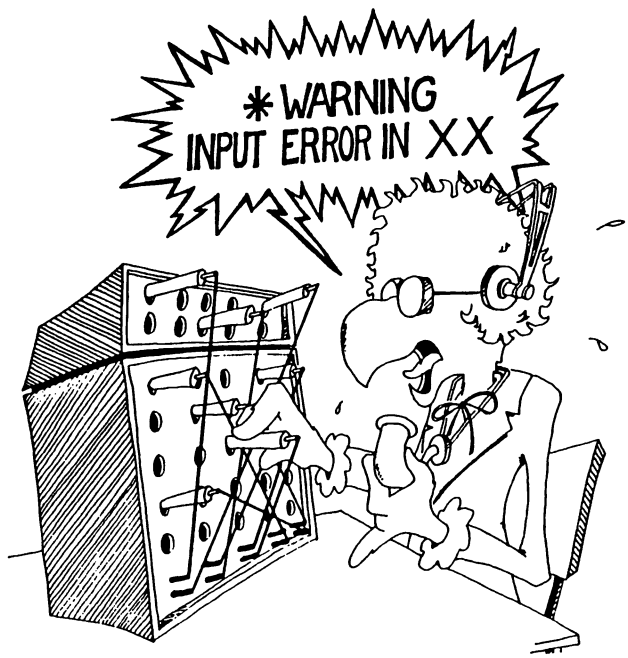
The computer answered:

```
* WARNING:
  INPUT ERROR IN XX
TRY AGAIN:
```

You should type your answer again, but not have any commas in it.

Rule: Do not put any commas into the string you type in answer to INPUT, unless ...

YOU REALLY WANT TO USE COMMAS!



Run the program again and answer the question with:

```
"HI, THERE"
```

This time the computer works OK, no nasty messages.

Rule: Put quotation marks around the answer if it has commas in it.

Assignment 5:

1. Write a program which asks for a person's name and then says something silly to the person by name.
2. Write a program which asks you to INPUT your favorite color and put it into a box called C$. Now the program asks you to INPUT your favorite animal and also puts this into box C$. Now tell the program to print C$. What will be printed? Run the program and see if you are right.

INSTRUCTOR NOTES 6 TRICKS WITH PRINT

In this lesson:

PRINT with a semicolon at the end
PRINT with semicolons between items
the "invisible" PRINT cursor
the INS key

The use of commas in PRINT is ignored as it is of little use on a 32 column screen.

The lesson introduces the output cursor which is invisible on the screen. It marks the place where the next character will be placed on the screen by a PRINT command. (The input cursor is the flashing square. It is familiar from the command mode and the INPUT command.)

When a PRINT statement ends with a semicolon, the output cursor remains in place and the next PRINT will put its first character exactly in the spot following the last character printed by the current PRINT command.

Without a semicolon at the end, the PRINT command will advance the output cursor to the beginning of the next line as its last official act.

A PRINT command can print several items, a mixture of string and numerical constants, variables, and expressions. Numerical constants and variables have not yet been introduced. The items are separated by semicolons.

The series of printed items will have their characters in contact. If spaces are desired, as in the "HAM AND EGGS" example, the spaces have to be put into the strings explicitly.

QUESTIONS:

1. Which cursor is a little flashing square? What command puts it on the screen?
2. Which cursor is invisible? What command uses it?
3. How do you make two PRINT statements print on the same line?
4. Will these two words have a space between them when run?

```
10 PRINT "HI";"THERE!"
```

   If not, how do you put a space between them?

LESSON 6 TRICKS WITH PRINT

ONE LINE OR MANY?

Enter this program:

```
10 REM FOOD
20 PRINT
30 PRINT "HAM"
40 PRINT "AND"
50 PRINT "EGGS"
```

and run it. Each PRINT command prints a separate line.

Now enter:

```
30 PRINT "HAM ";
40 PRINT "AND ";
```

(Don't change or erase the other lines.)

Be careful to put the space at the end of "HAM" and at the end of "AND" and the semicolon at the end of each line.

Run it.

What was different from the first time?

.....

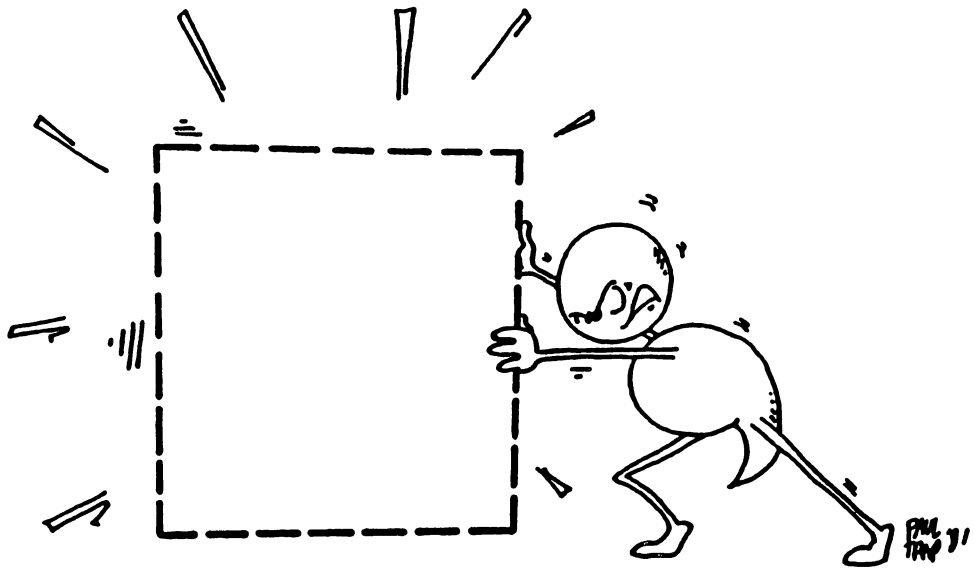
.....

THE HIDDEN CURSOR

Remember the flashing square? It is the INPUT cursor. It shows where the next letter will appear on the screen when you type.

The PRINT command also has a cursor, but it is invisible. It marks where the next letter will appear when the computer is PRINTing.

Rule: The semicolon makes the invisible PRINT cursor wait in place on the screen. The next PRINT command adds on to the same line.



FAMOUS PAIRS

Enter:

```
10 CALL CLEAR
20 PRINT "ENTER A NAME"
30 ENTER A$
40 PRINT "ENTER ANOTHER"
50 ENTER B$
60 CALL CLEAR
70 PRINT"PRESENTING THAT FAMOUS TWOSOME"
75 PRINT
80 PRINT A$;" AND ";B$
```

Be sure to put a space before and after the "AND".

SQUASHED TOGETHER OR SPREAD OUT?

Enter NEW then try this:

```
10 PRINT "ROCK";"AND";"ROLL"
```

after you have run it, try also:

```
10 PRINT "ROCK ";"AND ";"ROLL"
```

don't forget the spaces after ROCK and AND.

THE INSERT KEY

Type this:

```
10 REM MAGIC KY
```
(do not press ENTER yet)

We left the "E" out of KEY!

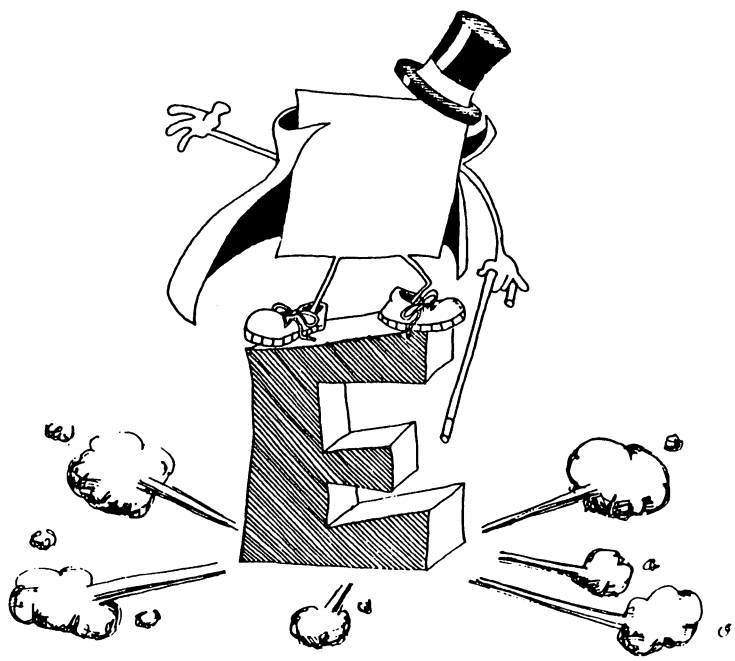
Use the FCTN and arrow keys to put the cursor over the "Y".

Hold the FCTN key down and press the INS key.

(INS stands for INSERT. "Insert" means "stick in.")

Now press the "E" key. An "E" pops in front of the "Y". Press ENTER.

Rule: The letter is inserted to the left of the flashing cursor.



You can insert more than one letter at the same spot if you want.

Fix this: `10 REM DRON` (make it "DRAGON")

To stop inserting, just use the FCTN and an arrow key to move the cursor somewhere. After that, any letter you type will erase the letter the cursor is on.

Assignment 6:

1. Type these lines and then fix them using the INS key.

```
10 REM   WIZRD       (WIZARD)
10 REM   TAS         (TEXAS)
10 REM   CMPTR       (COMPUTER)
```

2. Write a program which asks for the name of a musical group and one of their tunes. Then using just one PRINT command, print the group name and the tune name, with the word "plays" in between.
3. Do the same, but use 3 print commands to print on one line.

INSTRUCTOR NOTES 7 THE LET STATEMENT

The LET statement is introduced using the concept of memory boxes. Concatenation using the "&" symbol is called "gluing the strings."

The box model is used to emphasize that LET is a replacement command, not an "equal" relationship in the sense used in arithmetic.

The box idea nicely separates the concepts "name of the variable" and "value of the variable." The name is on the label of the box, the value is inside. The contents of the box may be removed for use and new contents inserted.

More exactly, a copy of the contents is made and used when a variable is used; the original contents remain intact. This point is explained.

Used so far:

NEW, PRINT, REM, RUN, CALL SOUND, CALL SCREEN, CALL CLEAR, LIST, INPUT, LET

Special keys discussed so far:

ENTER, SHIFT, FCTN, two arrow keys, DEL, INS.

QUESTIONS:

1. LET puts things in boxes. So does INPUT. How are they different?
2. In this program:

```
10 Q$="MOM"
```

   what is "MOM" called? What is the name of the string variable in this program? What is the value of the string variable after the program runs?
3. If you run this little program:

```
10 LET H$="FAT"
20 LET K$=" SAUSAGE"
30 LET P$=A$ & K$
```

   what is in each box after the program runs?

LESSON 7 THE LET STATEMENT

The LET command puts things into boxes. Enter and run:

```
10 CALL CLEAR
20 LET Q$="TRUCK"
40 PRINT Q$
```

Here is what the computer does:

Line 10 The computer clears the screen.

Line 20 It sees that a box named "Q$" is needed. It looks in its memory for it. It doesn't find one because "Q$" has not been used in this program before. So it takes an empty box and writes "Q$" on the front, and then puts the string "TRUCK" into it.

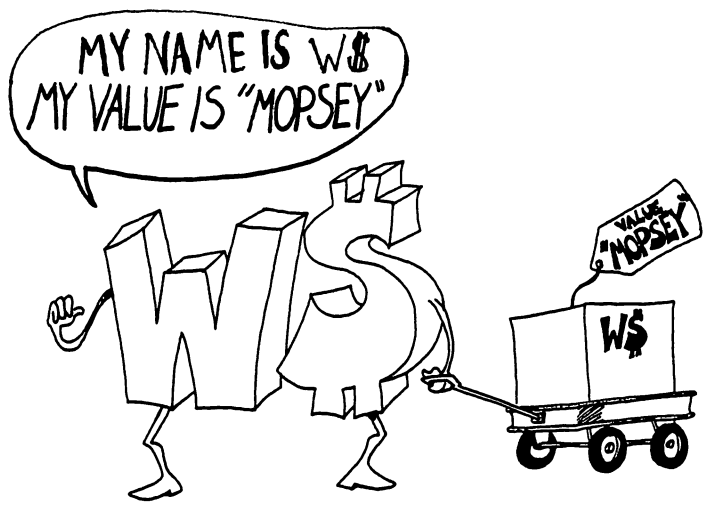
Line 40 The computer sees that it must print whatever is in box "Q$". It goes to the box and makes a copy of the string "TRUCK" that it finds there. It puts the copy on the TV screen. The string "TRUCK" is still in box "Q$".

NAMES AND VALUES

This line makes a string variable:

```
30 W$="MOPSEY"
```

The name of the variable is W$. The value of the variable is put into the box. In this line the value of W$ is "MOPSEY"



ANOTHER EXAMPLE:

Enter and run:

```
10 LET D$="PICKLES"
20 LET A$=" AND "
30 PRINT "WHAT GOES WITH PICKLES?"
35 INPUT Z$
40 CALL CLEAR
50 PRINT D$;A$;Z$
```

Explain what the computer does in each line.

10 .....

20.....

30.....

40.....

50.....

GLUING THE STRINGS

Here is how to stick two strings together to make a longer string. Enter:

```
10 CALL CLEAR
20 LET W$="HAR DE "
25 LET X$="HAR "
30 L$=W$ & X$
40 PRINT L$
50 PRINT
60 LET L$=L$ & X$
70 PRINT L$
```

Before you RUN this program, try to guess what will be printed at line 40 and at line 70:

40 .....

70 .....

Now run the program to see if you were right.

Rule: The "&" sign sticks two strings together.

The "&" sign is called "Ampersand." It means the "and" sign of Mr. Amper.

Assignment 7:

1. Write your own program which uses the LET command and explain how it stores things in "boxes."
2. Write a program which inputs two strings, glues them together and then prints them.

INSTRUCTOR NOTES 8 THE GOTO STATEMENT

The GOTO command allows a "dumb" loop that goes on forever. It also helps in flow of command in later programs, after the IF is introduced. It provides a slow and easy entrance into the idea that the flow of command need not just go down the list of numbered lines.

For now its main use is to let programs run on for a reasonable length of time. In each loop through, something can be modified.

In particular, a last statement like:

```
90 GOTO 90
```

keeps the program in the RUN mode (with the screen color specified by the CALL SCREEN( ) command) rather than falling back into the command mode.

The problem is how to stop it. The FCTN CLEAR keys do this.

The GOTO statement is like a knife in the hands of your student. If she does not learn to control its use, it can slice up her programs into spaghetti. Later lessons will show how to write program control structures like DO UNTIL and DO WHILE that make proper use of the GOTO.

We now have three of the four major elements that lead to "real" programming. They are PRINT, INPUT and GOTO. Lacking is the IF, which will change the computer from some sort of a record player into a machine which can evaluate situations and make decisions accordingly.

QUESTIONS:

1. Look at this little program:

```
10 PRINT "HI"
20 GOTO 40
30 PRINT "BIG"
40 PRINT "DADDY"
```

   What will you see on the screen when it is run?

2. And this one:

```
10 PRINT "DRAGON ";
20 PRINT "DUST ";
25 CALL SOUND(100,110,10)
30 GOTO 20
```

3. How do you stop the program in question 2?

4. Write a short program which "peeps," asks you your favorite movie star's name, and then does it over and over again.

LESSON 8 THE GOTO STATEMENT

THE KEYBOARD OVERLAY STRIP

The strip overlay is a narrow piece of plastic which has words written along it:

DEL INS ERASE CLEAR BEGIN PROC'D AID REDO BACK QUIT

It belongs in the slot above the top row of keys.

We show how to use these words on the strip:

word	key	where explained
DEL	....1....	lesson 4
INS	.....	lesson 6
CLEAR	.....	this lesson
QUIT	.....	this lesson

Fill in which key is below each word.

The rest of the words are not used in TI BASIC.

JUMPING AROUND IN YOUR PROGRAM

Try this program:

```
10 CALL CLEAR
20 PRINT"YOUR NAME?"
25 INPUT N$
30 PRINT N$
35 PRINT
40 GOTO 30
```

RUN this program. It never stops by itself! To stop your name from whizzing past your eyes:

hold down the FCTN key

and

press the 4 key.

FCTN 4 means FCTN CLEAR. It stops the program and clears the screen.

Line 40 uses the GOTO command. It is like "GO TO JAIL" in a game of Monopoly. Every time the computer reaches line 40, it has to go back to line 30 and print your name again.

We will use GOTO in a lot of programs.

Warning!

It looks like FCTN QUIT (holding the FCTN key down and pressing the "equal sign" key) would stop the program from running.

It does, BUT ...

It also ERASES THE PROGRAM!

Always use FCTN CLEAR to stop the program from running.

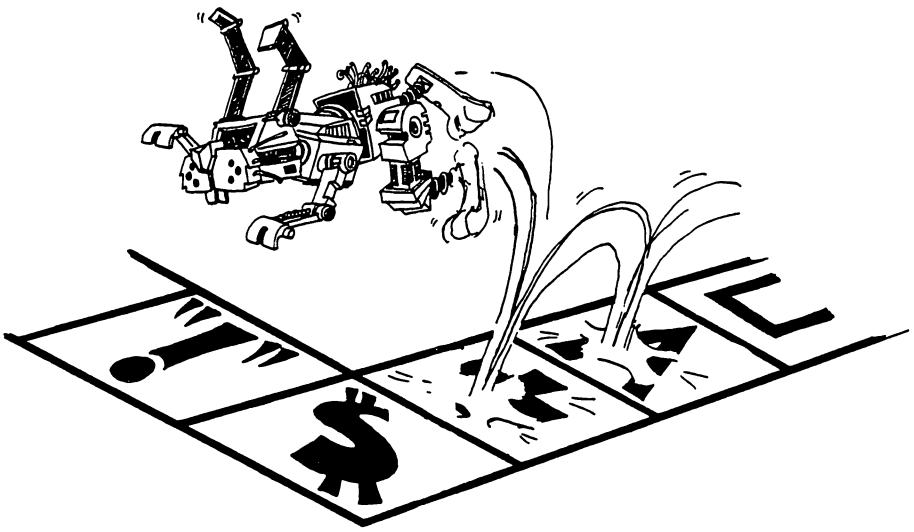
MORE JUMPING

Enter:

```
20 PRINT "SAY SOMETHING"
30 INPUT S$
35 PRINT
40 PRINT "DID YOU SAY '";S$;"'?"
45 PRINT
50 GOTO 30
```

Run the program. Type an answer every time you see the "?" and the flashing cursor. Press the FCTN CLEAR keys to end the program.

Notice the arrow from line 50 to line 30. It shows what the GOTO does. You may want to draw such arrows in your program listings.



KINDS OF JUMPS

There are only two ways to jump: ahead or back.

Jumping back gives a LOOP.

```
10 PRINT "HI"
20 GOTO 10
```

The path through the program is like this:

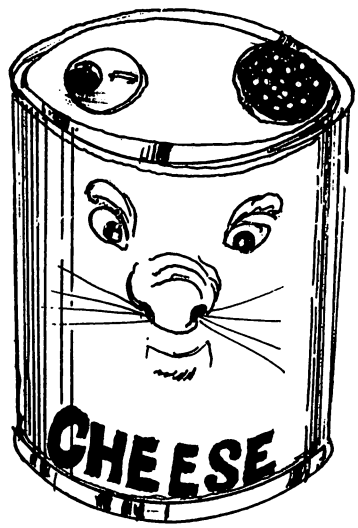
The computer goes around and around in this loop. Press the FCTN CLEAR keys to stop.

Jumping ahead lets you skip part of the program. It is not done as often as jumping back.

A CAN OF SPAGHETTI

Look at this:

```
START
10 REM ::: SPAGHETTI :::
20 GOTO 70
25 PRINT "A"
26 GOTO 50
30 PRINT "S"
31 GOTO 25
40 PRINT "C"
41 GOTO 90
50 PRINT "U"
51 GOTO 40
70 PRINT "SPAGHETTI"
71 GOTO 30
90 PRINT "E"
99 REM ::: END :::
WHEW!
```



This is NOT a good, clear program!

It is a "spaghetti program."

Don't write spaghetti programs! Don't jump around too much in your programs.

Assignment 8:

1. Just for practice in understanding the GOTO statement, draw the road map for this spaghetti program:

```
10 REM ::: FORKED TONGUE :::
20 GOTO 40
30 PRINT "N"
31 GOTO 60
40 PRINT "S"
41 GOTO 30
50 PRINT "E"
51 GOTO 99
60 PRINT "A"
61 GOTO 90
90 PRINT "K"
91 GOTO 50
99 PRINT "B I T E"
```

2. Write a program which prints "TEEN POWER" over and over.
3. How do you stop your program?
4. Write another which prints your name on one line, then a friend's on the next, over and over. Sound a tone as each name is printed. Stop the program with the FCTN CLEAR keys.
5. Write a program which uses each of these commands: CALL CLEAR, PRINT, INPUT, LET, GOTO. It also should glue two strings together.

INSTRUCTOR NOTES 9 TAB AND DELAY LOOPS

In this lesson: TAB, function arguments, delay loops.

Delay loops slow the program down so that its operation can be more easily observed. They also are used for portions of the program which must run at certain speeds, and should then be called "timing loops."

The TAB command adds flexibility to the screen display.

TAB is used in a PRINT command and is like the tab on a typewriter. It allows interesting displays of verbal information.

Students who are not very familiar with a typewriter may need extra help in seeing what a TAB is good for.

Several TAB commands can be used in one PRINT statement, but the arguments in the ( ) must increase each time. That is, TAB cannot be used to move the cursor back to the left.

This lesson introduces loops in a painless way.

The delay loop is on two adjacent lines. The amount of delay is determined by the size of the loop variable. A value of 350 gives about a one second delay.

After seeing that the primary work of the loop is simply to count until a particular value is reached before going on to the next instruction, it will be easier for the student to handle loops in which things are going on inside.

QUESTIONS:

1. Show how to write a delay loop which lasts for about 2 seconds.
2. Will this work for a delay loop?

```
120 FOR Q=300 TO 500
122 NEXT Q
```

3. Tell what the computer will do in each case:

```
10 PRINT "HI";TAB(20);"GOOD LOOKING!"
10 TAB(5);PRINT "OH-OH!"
10 PRINT TAB(15);"NOPE";TAB(1);"NOT HERE"
```

Run each and see if you are right.

LESSON 9 TAB AND DELAY LOOPS

THE TAB COMMAND

TAB in a PRINT command is like the TAB on a typewriter. It moves the printing cursor to a new spot to the right.

(The printing cursor is invisible.)

The next thing to be printed goes where the cursor is.

Try this:

```
10 PRINT "123456789ABCDEF"
20 PRINT "Y";TAB(5);"Z"
```

Rule: After TAB(N), the next character will be printed in column N.

CAREFUL!

Run this:

```
5 TAB(5)
```

You see

```
INCORRECT STATEMENT
IN 5
```

TAB( ) has to be in a PRINT command. You cannot use TAB( ) by itself.

YOU CANNOT TAB BACKWARDS

Try this:

```
10 PRINT "123456789ABCDEF"
20 PRINT "1";TAB(9);"9";TAB(3);"3"
```

The TAB( ) command can move the printing only to the right.

If you try to move back to the left, the computer moves down one line first.

YOUR NAME IS FALLING!

```
10 CALL CLEAR
15 LET N=1
20 PRINT"YOUR FIRST NAME"
30 INPUT W$
40 PRINT TAB(N);W$
50 LET N=N+1
60 GO TO 40
```

Press FCTN CLEAR to stop the run.

This program prints your name in a diagonal down the screen, top left to bottom right. Try other values of N. Try changing lines:

```
15 LET N=25
50 LET N=N-1
```

HOW BIG A SPACE CAN TAB( ) MAKE?

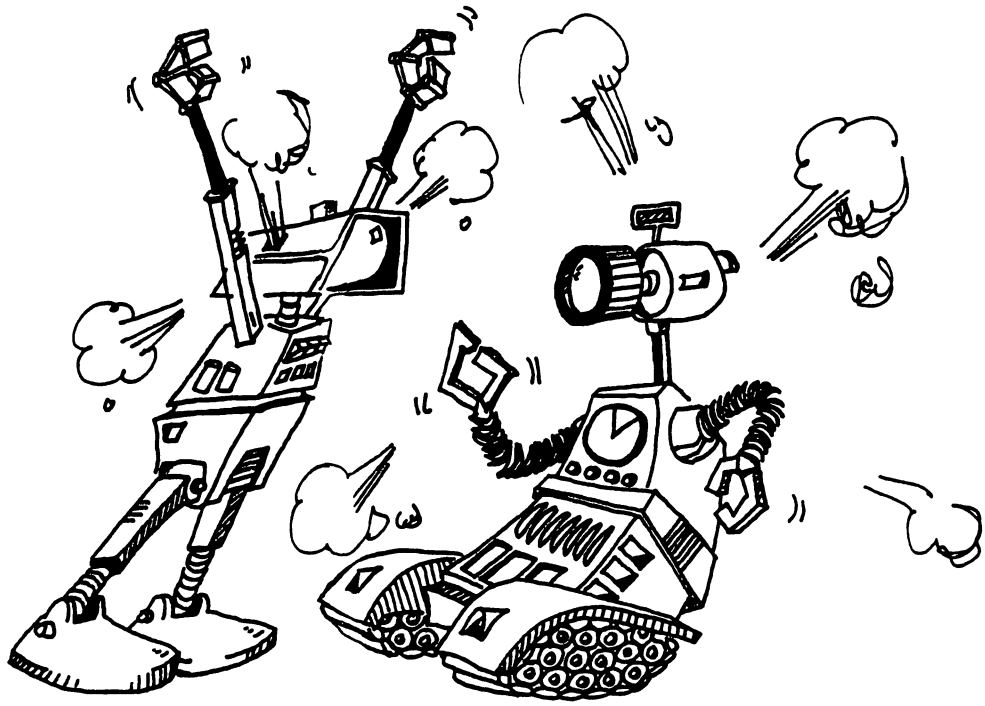
There are 28 spaces across the screen. You can use any number 1 through 28 inside the TAB( ) parentheses. Larger numbers make the computer skip lines.

FUNCTIONS DON'T FIGHT BUT THEY HAVE ARGUMENTS

TAB( ) is a command which is like a "function."

We will study other functions like RND( ), INT( ), SEG$( ), etc.

The number inside the ( ) is called "the argument of the function." TAB( ) says "move the cursor over" and the argument tells "where to move it."



Assignment 9A:

1. Write a program which asks for last names and nicknames. Then print the last name starting at column 5 and the nickname at column 15. Use a GOTO so the program is ready for another nick-name pair.
2. Write an "insult" program. It asks your name. Then it peeps and writes your name. Then it TABS over in the line and prints an insult.

INSTRUCTOR NOTES 10 INTRODUCING NUMBERS

Numerical variables and operations are introduced. The LET, INPUT and PRINT commands are revisited.

The idea of memory as a shelf of "boxes" is extended to numbers. Again, variable names are limited to one letter for the time being.

The arithmetic operations are illustrated. The "*" symbol for multiplication will probably be unfamiliar to the student. Division will give decimal numbers, so it is nice if your student is familiar with them. But most arithmetic will be addition and subtraction, with a little multiplication, and a student unfamiliar with decimal numbers will not experience any disadvantage.

It may seem strange to the student that the numbers in string constants are not "numbers" which can be used directly in arithmetic. The VAL and STR$ functions will be introduced later in the book and allow interconversion of numbers and strings.

A mixture of string and numerical values can be printed by PRINT.

The non-standard use of "=" in BASIC, that it means "replace" and not "equal," shows up strongly in the statement:

```
LET N=N+1
```

The cartoon uses the box idea to illustrate this meaning of "=".

QUESTIONS:

1. What are the three kinds of "boxes" in memory? (That is, named by the kinds of things stored in the boxes.)
2. Explain why "N = N + 1" for a computer is not like "4 = 4 + 1" in arithmetic.
3. Give another example of "bad arithmetic" in a LET command. Use the *, or / symbols.
4. What does the computer mean by "STRING-NUMBER MISMATCH"?
5. Give an example of a program line which would have a STRING-NUMBER MISMATCH.
6. Explain what is meant by the "name of a variable" and the "value of a variable" for numerical variables. For string variables.

LESSON 10 INTRODUCING NUMBERS

INPUT, LET AND PRINT

So far we have used only strings. Numbers can be used too. Enter and run this program:

```
10 REM BIGGER
15 CALL CLEAR
20 PRINT"GIVE ME A NUMBER"
30 INPUT N
40 LET A=N+1
50 PRINT"HERE IS A BIGGER ONE"
60 PRINT A
```

ARITHMETIC

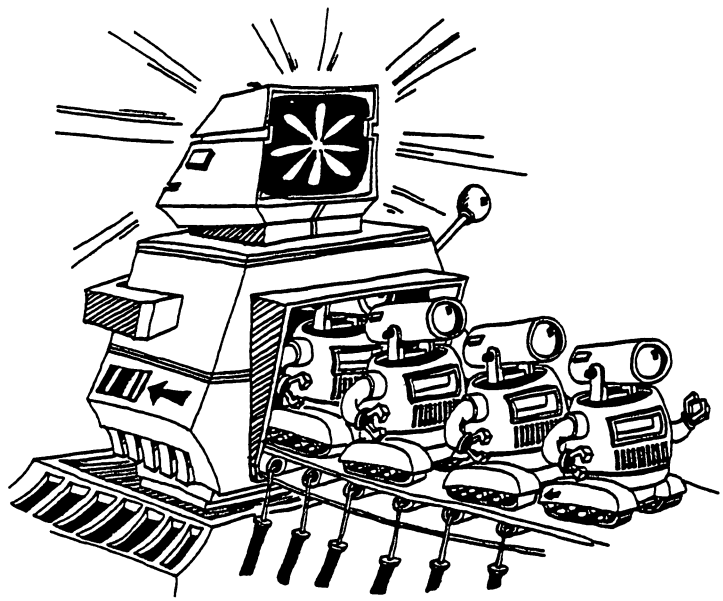
	symbol	key
addition	+	SHIFT =
subtraction	–	SHIFT /
multiplication	*	SHIFT 8
division	/	/

Computers use "*" instead of "×" for a multiplication sign.

Try this. Change line 40 so that N is multiplied by 5.

Computers use "/" for a division sign. Answers are given as decimals.

Try this: Change line 40 so that A is N divided by 5. What do you say in line 50?



DELAY LOOPS

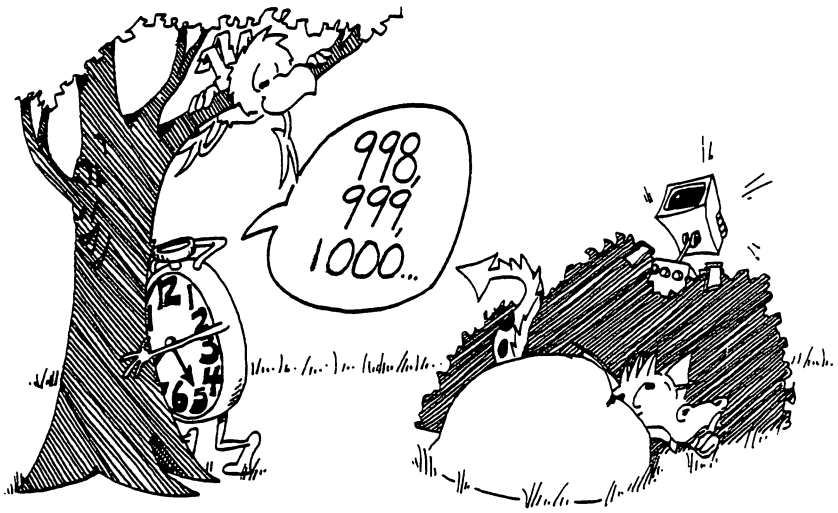
Here is a way to slow down the program. It is a "delay loop."

Run this program:

```
10 REM HIDE AND SEEK
20 CALL CLEAR
30 PRINT "YOU ARE IT"
40 FOR T=1 TO 1000
41 NEXT T
50 PRINT "COMING READY OR NOT"
```

Lines 40 and 41 are the delay loop. The computer counts from 1 to 1000 before going on to the next line. It is like counting when you are "it" in a game of hide and seek.

Try changing the number "1000" in line 40 to some other number.



Each 350 in the delay loop is worth about 1 second of time. Try this:

```
10 REM ---- TICK TOCK ----
20 CALL CLEAR
30 PRINT "WAIT HOW LONG? ";
31 INPUT S
36 T=S*350
40 FOR I=1 TO T
41 NEXT I
45 PRINT
46 SOUND(500,300,10)
50 PRINT S;" SECONDS ARE UP"
```

Line 36 has a multiplication in it. We will study this in Lesson 10.

Assignment 9B:

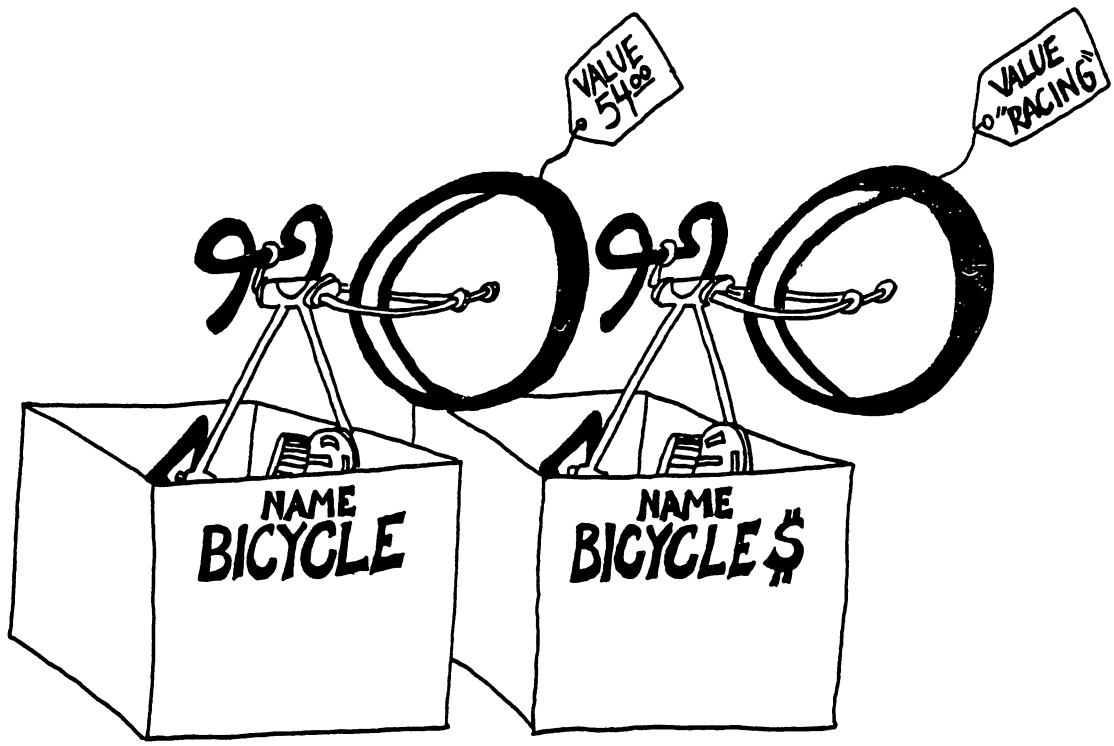
1. Write a "slow poke" program. Make it print "STEP BY STEP" with several seconds between each word. Have the computer peep before each word.

VARIABLES

The name of a box that contains a string must end with a dollar sign. Examples: N$, A$, Z$.

The name of a box that contains a number doesn't have a dollar sign. Examples: N, A, Z.

The thing which is put into the box is called the "value" of the variable.



ARITHMETIC IN THE LET COMMAND

```
10 LET A=2
20 LET B=3
30 LET C=B-A
40 PRINT A;B;C
```

Some more examples:

```
10 LET B=15
20 LET A=B/5
30 LET X=A*4+2
40 PRINT X;A
```

CAREFUL!

Numbers and strings are different. Example: "1984" is not a number. It is a string constant because it is in quotes.

Rule: Even if a string is made up of number characters it is still not a number.

Some numeric constants: `5, 22, 3.14, -50`

Some string constants: `"HI", "7", "TWO", "3.14"`

Rule: You cannot do arithmetic with the numbers in strings.

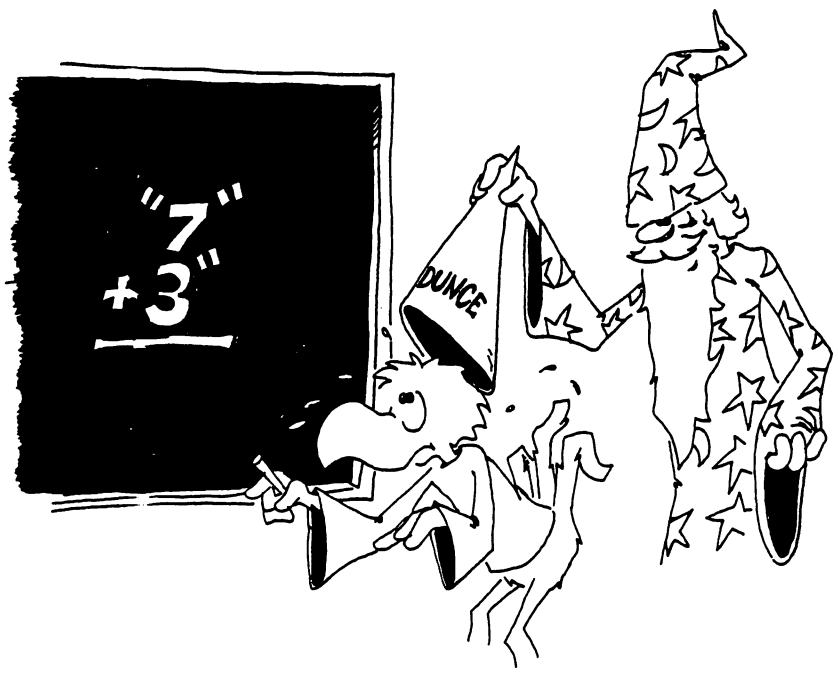
Correct:
```
10 LET A = 7 + 3
```

Wrong:
```
10 LET A$ = 7 + 3
```

Wrong:
```
10 LET A = "7" + "3"
```

If you run either of these wrong lines, the computer will print:

```
STRING-NUMBER MISMATCH
 IN 10
```



The two types of variables are "string" and "numeric." You cannot mix them.

Enter:

```
10 LET A=5
20 LET B$="10"
30 LET C=A+B$
```

Lines 10 and 20 are OK, line 30 is wrong. What will the computer do when you run this little program? ..... Try it.

Try to guess what each of these statements will print, then enter the line to see what happens:

```
PRINT 5          .....
PRINT "5"        .....
PRINT "5+3"      .....
PRINT "5"+"3"    .....
PRINT 5 + 3      .....
```

MIXTURES IN PRINT

You can print numbers and strings in the same PRINT command. (Just remember that you cannot do arithmetic with the mixture.)

Correct:

```
PRINT A;"SEVEN";" 7"

PRINT A;B$
```

Run this line.

```
10 PRINT 5/2;" IS EQUAL TO 5/2"
```

A FUNNY THING ABOUT THE EQUAL SIGN

The "=" sign in computing does not mean "equals" exactly. Look at this program:

```
10 LET N=N+1
```

This does not make sense in arithmetic. Suppose N is 7. This would say that:

```
7=7+1
```

which is not correct.

But it is OK in computing to say N = N + 1 because the "=" sign really means "replace." Here is what happens:

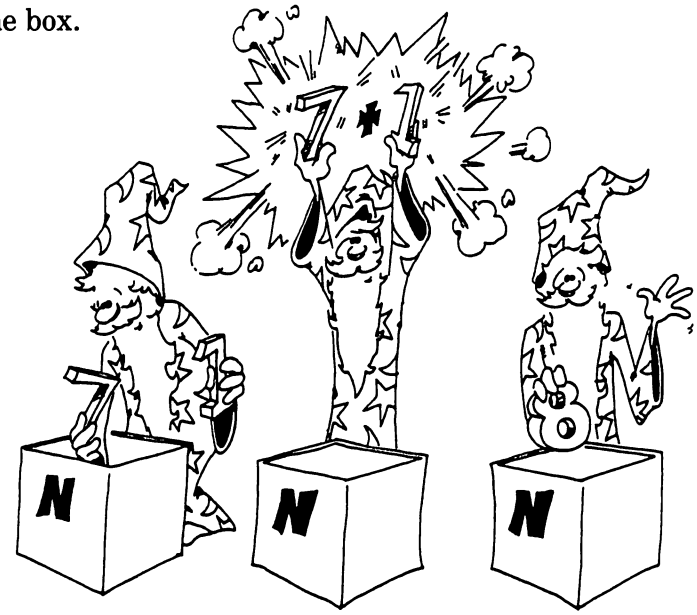
```
10 LET N=N+1
```

The computer goes to the box with N written on the front.

It takes the number 7 from the box.

It adds 1 to the 7 to get 8.

Then it puts the 8 in the box.



Another way to say the same thing is:

```
10 LET N=N+1      means
```

Let (old N) equal (new N) plus 1

NOT BACKWARDS

In arithmetic, these two equations mean the same thing:

N = 6
6 = N

But in computing you cannot put the LET statement backwards!

```
right:      30 LET N = 6
wrong:      30 LET 6 = N
```

Assignment 10:

1. Write a program which asks for your age and the current year. Then subtract and print out the year of your birth. Be sure to use PRINT statements to tell what is wanted and what the final number means.
2. Write a program which asks for two numbers and then prints out their product. (Multiplies them.)

INSTRUCTOR NOTES 11 FOR-NEXT LOOPS

FOR, NEXT and STEP key words which make loops are described in this lesson. Nested loops are explained.

The loop is made of two statements, one starting with FOR and the other with NEXT. These commands may be separated by several lines and yet are strongly interdependent. The delay loop in a previous lesson helps form the notion that the FOR... and the NEXT are coupled. It remains then to show the utility of repeating a set of commands in the middle of the loop.

Before the program starts to run, BASIC checks to see if the number of FOR statements matches the number of NEXT statements. If not, a FOR-NEXT ERROR message is printed.

Unlike most dialects of BASIC, TI BASIC checks whether the condition for exit is already satisfied before entering the loop. If it is satisfied, the loop is skipped.

The FOR statement is evaluated just once at the time the loop is entered. It puts the starting value of the loop variable into variable storage where it is treated just as any other numerical variable. (For example, the loop variable can be changed in the body of the loop.) The STEP value, the ending value, and the address of the first statement after the FOR are put on a stack.

From then on, all the looping action takes place at the NEXT command. Upon reaching NEXT, the loop variable is incremented by the value of the STEP and compared with the end value. If the loop variable is larger than the end value (or smaller in the case of negative STEPs) NEXT passes control to the statement after itself. Otherwise, it gives control to the statement after the FOR command.

Jumping into the middle of a loop is usually a disaster. Jumping out of a loop before reaching NEXT is commonly done, but in some cases (especially where subroutines are involved) may make hard to find bugs.

QUESTIONS:

1. Write a loop which prints the numbers from 0 to 20 by two's.
2. Write a "Ten Little Indians" program loop which prints from 10 down to zero Indians.
3. Write a pair of nested loops to print "MINI" in the outside loop and "HA" in the inside loop. Print 3 of the MINIs and for each of them print 2 of the HAs.

LESSON 11 FOR-NEXT LOOPS

Remember the delay loop? The computer counted from 1 to 1000 and then went on.

```
30 FOR T=1 TO 1000
31 NEXT T
```

The computer is smarter than that. It can do other things while it is counting.

Run this:

```
10 REM COUNTING
20 CALL CLEAR
30 FOR I=5 TO 20
40 PRINT I
50 NEXT I
```

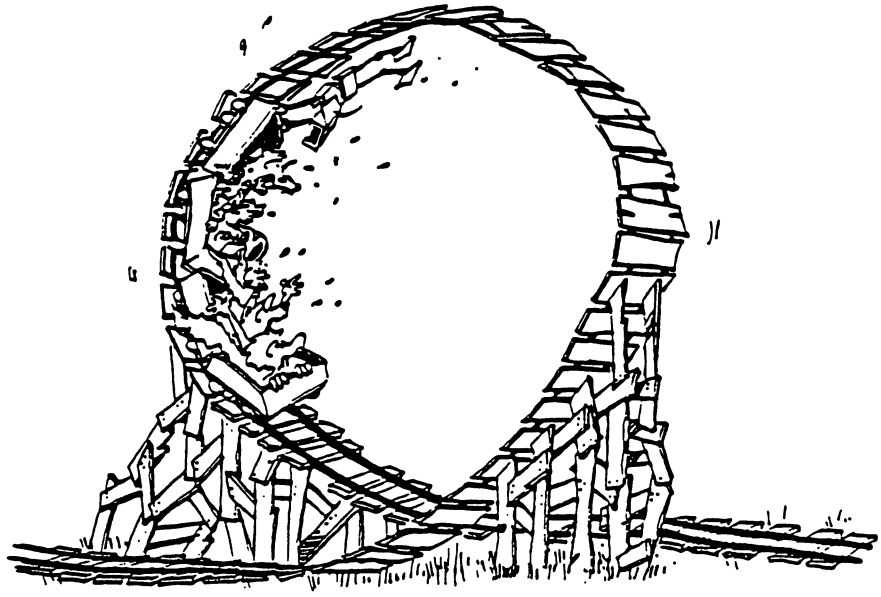
The loop can start on any number and end on any higher number. Try changing line 30 in these ways:

```
30 FOR I=100 TO 101
30 FOR I=-7 TO 13
30 FOR I=1.3 TO 5.7
```

MARK UP YOUR LISTINGS

Show where the loops are by arrows:

```
10 REM ROBIN HOOD
20 CALL CLEAR
30 FOR I=0 TO 7
40 PRINT I
50 NEXT I
```



THE STEP COMMAND

The computer was counting by one's in the above programs. To make it count by two's, change line 30 to this:

```
30 FOR I=10 TO 30 STEP 2
```

Assignment 11A:

1. Have the computer count by five's from zero to 100.

COUNT DOWN LOOPS

You can make the computer count down by using a negative STEP.

Try this:

```
10 REM *** APOLLO 11 ***
20 CALL CLEAR
30 PRINT "T MINUS 12 SECONDS AND COUNTING"
40 FOR I=11 TO 0 STEP -1
50 PRINT I
51 CALL SOUND(100,900,10)
60 REM ---------------------TIMING LOOP
61 FOR J=1 TO 350
62 NEXT J
70 NEXT I
71 CALL SOUND(1000,110,30)
75 PRINT "ALL ENGINES RUNNING. LIFT OFF."
76 CALL SOUND(1000,110,30)
80 PRINT "WE HAVE A LIFT OFF."
81 CALL SOUND(1000,110,30)
85 PRINT "32 MINUTES PAST THE HOUR."
86 CALL SOUND(1000,110,30)
90 PRINT "LIFT OFF ON APOLLO 11."
```

Lines 61 and 62 are the timing loop.

Lines 71, 76, 81, 86 use the SOUND command to make a delay too. The sound is turned off by making the "loudness" equal to 30.

NESTED LOOPS

In this program, we have one loop inside another.

The outside loop starts in line 40 and ends in line 70.

The inside loop starts in line 61 and ends in line 62.

These are "nested loops." It is like the baby's set of toy boxes which fit inside each other.

LOOP VARIABLES

To make sure that each FOR command knows which NEXT command belongs to it, the NEXT command ends in the "loop variable" name. Look at lines 61 and 62:

```
61 FOR J=1 TO 350
62 NEXT J
```

J is the loop variable. And for the loop starting in line 40:

```
40 FOR I=12 TO 0 STEP -1
          ...
70 NEXT I
```

I is the loop variable.

BADLY NESTED LOOPS

The inside loop must be all the way inside:

Right:

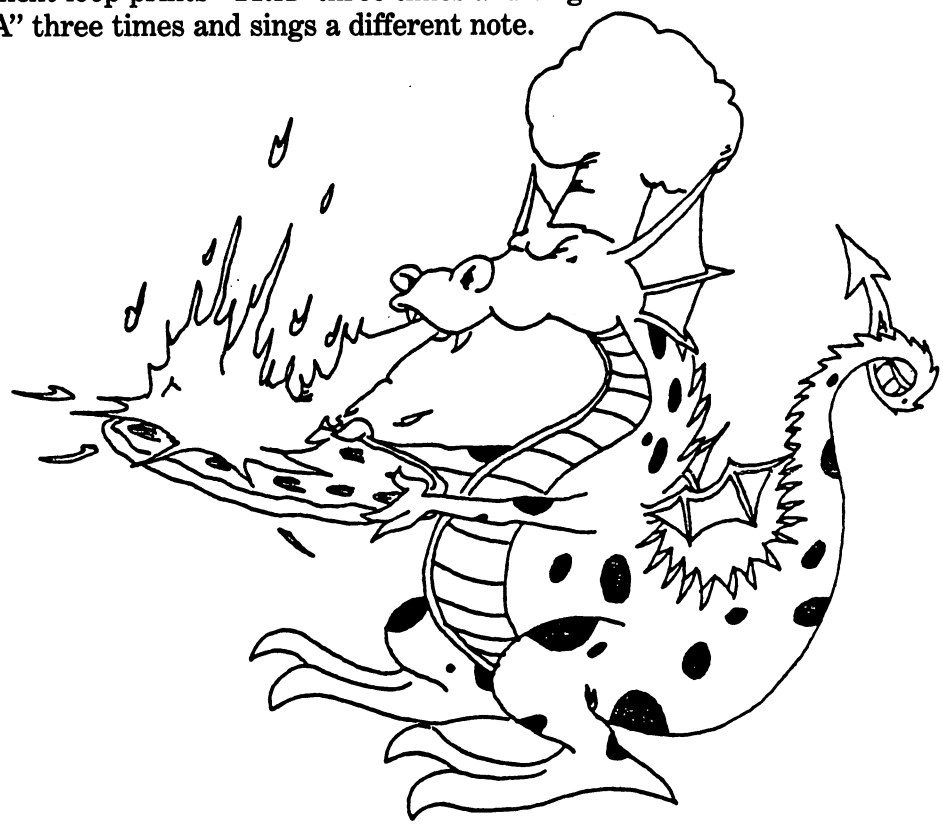
```
25 FOR X=3 TO 7
30 FOR Y=3 TO 7
40 PRINT X*Y
50 NEXT Y
60 NEXT X
```

Wrong:

```
25 FOR X=3 TO 7
30 FOR Y=3 TO 7
40 PRINT X*Y
50 NEXT X
60 NEXT Y
```

Assignment 11B:

1. Write a program which prints your name 15 times.
2. Now make it indent each time by 2 spaces more. It will go diagonally up the screen. Use TAB in a loop.
3. Now make it write your name on one line, your friend's name on the next and keep switching until each name is written 5 times.
4. Write a program with loops nested three deep. Do the outer loop three times and have it print "SING". Make it change the screen color on each pass through the loop. The next loop prints "TRA" three times and sings a note. The innermost loop prints "LA" three times and sings a different note.



INSTRUCTOR NOTES 12 RANDOM NUMBERS AND THE INT FUNCTION

This lesson introduces two functions: RND and INT. These are very important in games and also handy in making interesting displays like kaleidoscopes.

The RND function produces psuedo-random decimal numbers between 0.0 and 1.0. Such numbers are directly usable as probabilities, but integers over some range such as 1 to 6 for a die, or 1 to 13 for a suit of cards are often more to the point.

Your student may be shaky in decimal arithmetic, but all that is required here is multiplication of the random number by an integer, and perhaps also addition to an integer. The computer does the multiplication, of course, so only a rough idea of the desired result is necessary.

After extending the random number to a larger range than 0 to 1, conversion to an integer is desired. The INT function does this simply by truncating the number, "throwing away the decimal part." (For negative numbers the situation is a little more complicated, and that rare case is not treated here.)

The concept of functions is again used in this lesson and is further clarified.

The nesting of one function inside the parentheses of another is illustrated by using RND in the argument of an INT function.

QUESTIONS:

1. Tell what the computer will print for each case:

```
10 PRINT INT(G)
```

   and the box G contains:

```
2, 2.1, 2.95, 3.001, 67, 0, 0.2
```

2. Tell how the INT( ) function is different from "rounding off" numbers. Which is easier for you to do?
3. Tell how to change a number so that the INT( ) function will round it off.
4. What does the RND function do?
5. How can you get random integers (whole numbers) from 0 through 10? (Hint: INT(RND*10) is not quite right.)
6. How can you get random integers from 5 through 8?

LESSON 12 RANDOM NUMBERS AND THE INT FUNCTION

THE RND FUNCTION

When you throw dice, you can't predict what numbers will come up.

When dealing cards, you can't predict what cards each person will get.

The computer needs some way to let you "roll dice" and "deal cards" and do many other unpredictable things.

Use the RND function to do this. RND stands for "random."

Run this program:

```
10 REM RANDOM NUMBERS
15 CALL CLEAR
20 FOR I = 1 TO 20
25 LET N=RND
30 PRINT N
40 NEXT I
```

You see a lot of decimal numbers on the screen. The RND function in line 25 made them.

RND gives numbers which are decimals larger than 0 but smaller than 1. To make numbers larger than one, you just multiply.

Change the program above to:

```
25 LET N=RND*52
```

and run it again.

Now the numbers are between 0 and 52 in size. They could be used for choosing the 52 cards in a deck.

But:

We usually want whole numbers like 7 and 23 rather than decimal numbers like 7.03 and 23.62. Do this by using the INT function.

THE INT FUNCTION

"INT" stands for "integer" which means "whole number."

The INT function takes the number in its parentheses and throws away the decimal part, leaving an integer.
Try the INT function in this little program:

```
10 LET I=INT (6.3)
20 PRINT I
```

And in this:

```
10 LET X=0.3
20 PRINT "X= ";X;TAB(10);"INT(X)=";INT(X)
```

And this:

```
10 LET X=.3
20 LET Y=2.5
30 LET P=X+Y
40 LET Q=INT(X+Y)
50 PRINT P;Q
```

Look at the answers to see that the decimal part was thrown away.

Try this:

```
10 REM ----- INT -----
20 CALL CLEAR
30 PRINT"GIVE ME A DECIMAL NUMBER "
32 INPUT D
35 LET I=INT(D)
36 PRINT
40 PRINT "DECIMAL ";D;TAB(15);"INTEGER ";I
41 PRINT
42 PRINT
50 GOTO 30
```

ROLLING THE BONES

Usually dice games use two dice. One of them is called a "die." Here is a program which rolls a single die:

```
10 REM ////// ONE DIE //////
20 CALL CLEAR
30 LET R=RND
40 PRINT "RANDOM NUMBER";TAB(15);R
50 LET S=R*6
55 PRINT "TIMES 6";TAB(15);S
60 LET I=INT(S)
65 PRINT "INTEGER PART";TAB(15);I
70 LET D=I+1
75 PRINT "DIE SHOWS";TAB(15);D
77 PRINT
80 PRINT
82 FOR T=1 TO 2000
83 NEXT I
85 GOTO 20
```

WHAT GOES INSIDE THE ( ) ?

Numbers:

```
10 LET X=INT(34.7)
```

Variables:

```
10 LET X=INT(J)
```

Expressions:

```
10 LET X=INT(3*Y+2)
```

Functions:

```
10 LET X=INT(RND)
```

Here is how to save a lot of room.

Instead of:

```
30 LET R=RND
50 LET S=R*6
60 LET I=INT(S)70
```

Use just:

```
70 LET D=1+INT(RND*6)
```

EVERY PROGRAM RUN IS DIFFERENT

Each time you run a program, you want different cards or dice to show. Suppose your die rolls 5 on the first roll after starting the program. If you stop the program with FCTN CLEAR keys and then start it again, you want a different number than 5 to show (usually).

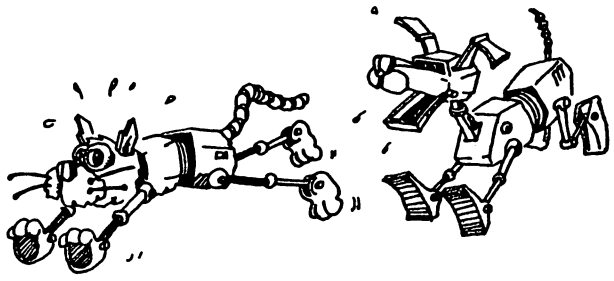
But you will always get the same number unless you do something about it! Command RANDOMIZE early in the program to make the computer choose a different random number than on the last run. Example:

```
10 REM MIXED UP
20 RANDOMIZE
30 PRINT INT(RND*100)
```

Run the program several times without line 20, then several times with line 20

Assignment 12:

1. Write a program which "rolls" two dice, called D1 and D2. Show the number on D1 and on D2 and the sum of the dice. You do not need the variables R, S, and I in the program above. They were used to show how the final answer was found.
2. Write a "paper, scissors, and rock" game - you against the computer. (Paper wraps rock, rock breaks scissors, scissors cut paper.) The computer chooses a number 1, 2 or 3 using the RND( ) function: 1 is paper, 2 is rock, 3 is scissors. You INPUT your choice as P, R, or S and the computer figures out who won and keeps score.



INSTRUCTOR NOTES 13 THE IF STATEMENT

The IF statement is a powerful but intricate command which is at the very heart of the computer as a logic machine.

The IF statement appeals both to our verbal and our visual imagination. The "cake" cartoon and the "fork in the road" cartoon illustrate these ideas. The GOTO command has already introduced the idea that the flow of control down the program may be altered. To that idea is now added the conditional test: if an "assertion" is true, a branch occurs; if it is false, program control continues to flow down the line list.

The assertion being tested for truth is called "phrase A." A jump to a new line number is made if the assertion is true.

Two levels of abstraction occur in the assertions. On the literal level we have "equal and not equal."

```
A$ = B$
C$ <> D$
```

The next level up we have the TRUTH or FALSITY of the assertion.

A two step process is needed to use the assertion properly:

1) What does the assertion say?

2) Is the assertion true?

The larger set of relations:

```
<    >    =    =<    =>    <>
```

will be treated later.

QUESTIONS:

1. How do you make this program print "THAT'S FINE"?

```
15 PRINT "DOES YOUR TOE HURT?"
17 INPUT T$
20 IF T$="NAH" THEN 90
40 GOTO 15
90 PRINT "THAT'S FINE"
```

2. Write a short program which asks if you like chocolate or vanilla ice cream. Answers to be "C" or "V". For the "C" print "Yummy!" For the "V" answer, print "Mmmmmm!"

LESSON 13 THE IF STATEMENT

Clear the memory and enter

```
10 CALL CLEAR
15 PRINT "HOLD YOUR BREATH "
20 PRINT "STILL HOLDING? (YES OR NO)"
30 INPUT A$
40 IF A$="YES" THEN 15
50 PRINT "WHY ARE YOU WHEEZING? "
```

Run the program. Try answering "YES", "NO" or "MAYBE". What happens?

YES ----------

NO ----------

MAYBE ----------

THE IF STATEMENT

The IF statement has two parts:

	40 IF	phrase A	THEN	number	N
It means:	40 IF	phrase A is true	THEN	go to line number	N
Example:	40 IF	A$="YES"	THEN		15

If A$="YES" is true, jump to line 15.

If A$="YES" is false, go to the next line.

CAREFUL! The "number N" must be a number. It cannot be a variable.

Right: `30 IF B$="HI" THEN 25`

Wrong: `30 IF B$="HI" THEN N`

Assignment 13A:

1. Write a program which asks if you are HAPPY. If you answer NO, it asks again. If you answer YES, it says GOOD.

THE "IF" IN ENGLISH AND IN BASIC

In English:

IF your homework is done, THEN you may have some cake.

In BASIC:

```
10 IF W$="DONE" THEN 30
20 PRINT "NOT DONE?"
25 PRINT "DO IT! THEN, "
30 PRINT "YOU MAY HAVE SOME CAKE"
```

The computer looks in box W$.

It sees if the string in the box is the same as "DONE". If it is, the program goes to line 30. If it is not, the program goes to the next line (20).

DRAWING MAPS IN YOUR PROGRAM

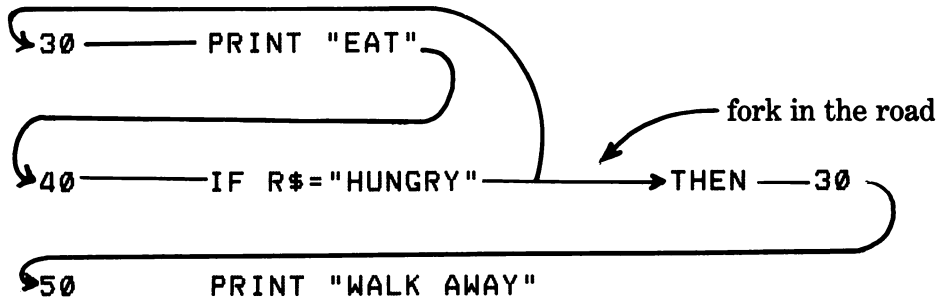
IF makes a fork in the road.

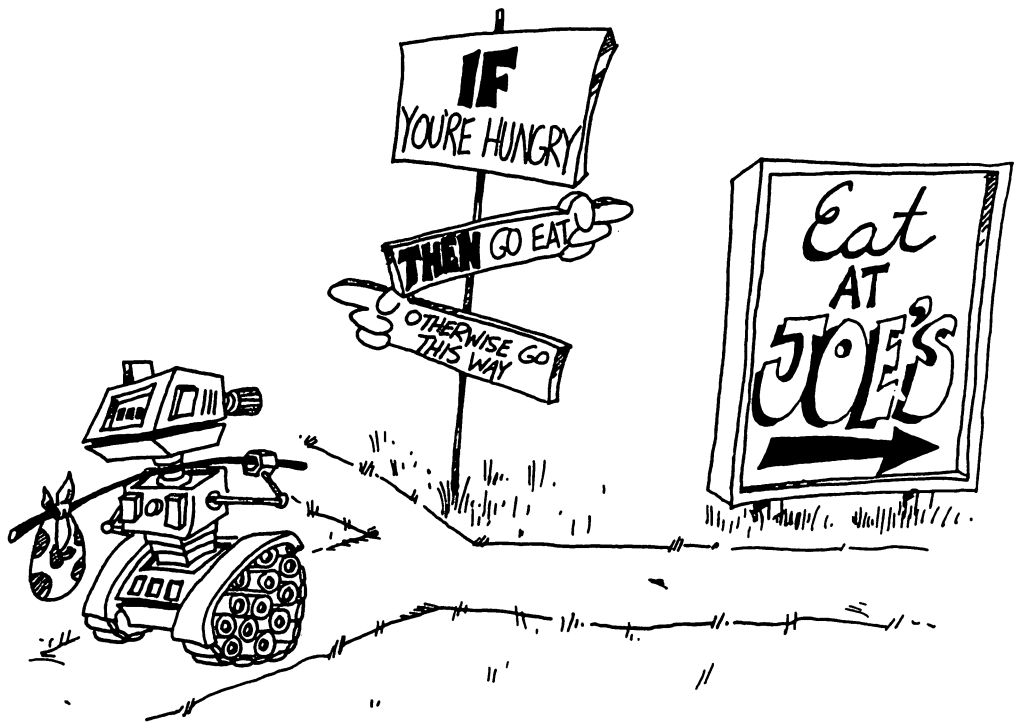
When it sees "IF" the computer must choose which road to take.

If "phrase A" is true, it must go to a new line number.

If "phrase A" is false, it goes down to the next line right away.

Here is the road map.





SKIPPING LINES

The IF command can skip ahead too.

Enter:
```
10 REM SKIP LINES
20 PRINT "ARE YOU HAPPY? <Y OR N>"
25 INPUT A$
30 REM ---------------------- IS PHRASE A TRUE?
31 IF A$="Y" THEN 60
40 REM ---------------------- THE ANSWER WAS "N"
45 PRINT "TOO BAD!"
47 GOTO 90
60 REM ---------------------- THE ANSWER WAS "Y"
65 PRINT "I'M GLAD"
90 REM THAT'S ALL FOLKS!
```

Assignment 13B:

1. Write a boy-girl program. Ask if the user is a "BOY" or a "GIRL". If the answer is "BOY" print "SNIPS AND SNAILS". If the answer is "GIRL", print "SUGAR AND SPICE".

THE "NOT EQUAL" SIGN

Two signs:

=	means	"equal"
<>	means	"not equal"

To make the "<>" sign:

hold down the SHIFT key
then press the "<" key, then the ">" key.

Look: `40 IF phrase A THEN 60`

"Phrase A" is a phrase that is TRUE or FALSE.

Pick `B$<>"COLD"` for "phrase A" and put it into:

```
40 IF B$<>"FIRE" THEN 60
```

Look:
If the B$ box contains "FIRE"
then B$ is not equal to "COLD"
so the phrase B$<> "COLD" is TRUE.

The computer will go to line 60.

Or	If	the B$ box contains "COLD"
	then	B$ is equal to "COLD"
	so	the phrase B$<> "COLD" is FALSE.

The computer will go to the next line.

Here is how it looks in a program:

```
10 PRINT "WITH DOGS IT'S A COLD NOSE"
11 PRINT
12 PRINT "WITH DRAGONS, IT'S ..."
13 PRINT
15 PRINT "HOW'S YOUR DRAGON'S BREATH?"
16 PRINT
20 PRINT "(ENTER 'FIRE' OR 'COLD')"
30 INPUT B$
35 PRINT
40 IF B$<>"COLD" THEN 60
50 REM ---------------------- THE DRAGON NEEDS HELP!
55 PRINT "FEED HIM SOME HOT CHILI, THEN ..."
56 PRINT
60 REM ---------------------- THE DRAGON IS FINE"
65 PRINT "PAT HIM ON THE HEAD, BUT WATCH OUT!"
80 PRINT
85 PRINT "'NICE DRAGON'"
```

Assignment 13C:

1. Write a "pizza" program. Ask what topping is wanted. Make the computer answer something silly for each different choice. You can choose mushrooms, pepperoni, anchovies, green peppers, etc. You can also ask what size.
2. Write a color guessing game. One player INPUTs a color in string C$ and the other keeps INPUTing guesses in string G$. Use two IF lines, one with a "something A"

   ```
   G$<>C$
   ```

   for when the guess is wrong, and the other with an "=" sign for when the guess is right.

INSTRUCTOR NOTES 14 SAVING TO TAPE

This lesson shows how to save programs to tape and how to load them again.

The commands SAVE and OLD are introduced.

The only other commands used are:

NEW	REM
LIST	PRINT

This lesson can be used anytime after Lesson 3.

We put it this late in the book because most programs up to this point are relatively short and uninteresting, not worth saving. The process of programming was being emphasized, not the end result of useful programs.

However, your own judgement should prevail. You can insert this chapter at an earlier point in the flow of lessons so that your student can save some programs she is particularly proud of.

Ordinary audio tape is usually satisfactory for computer use. However, remember that a tiny imperfection can cause the tape to "drop a bit" and this makes the program wrong, or worse, unloadable!

You will not need long tapes. In fact, the special 10 minute data tapes you can buy at computer stores are inexpensive and convenient.

The TI 99/4A computer supports use of named files but this is not discussed in this book. Please refer to the TI USER'S REFERENCE GUIDE.

QUESTIONS:

1. What command tells the computer to save a program on tape?
2. What command tells the computer to load a program from tape into the computer?
3. About how long does it take to save a short program?
4. If a program is put onto tape, is it still in the computer's memory?

LESSON 14 SAVING TO TAPE

CONNECTING THE RECORDER

Follow the directions in your Texas Instruments TI-99/4A COMPUTER USER'S REFERENCE GUIDE.

The first time you use the recorder, someone must find the correct adjustment of its volume and treble controls. If this has already been done, the instructor will write the settings down on the lines below.

Otherwise, the instructor will follow the directions in the TI USER'S REFERENCE GUIDE for finding the correct settings and then write them down here.

.....

.....

.....

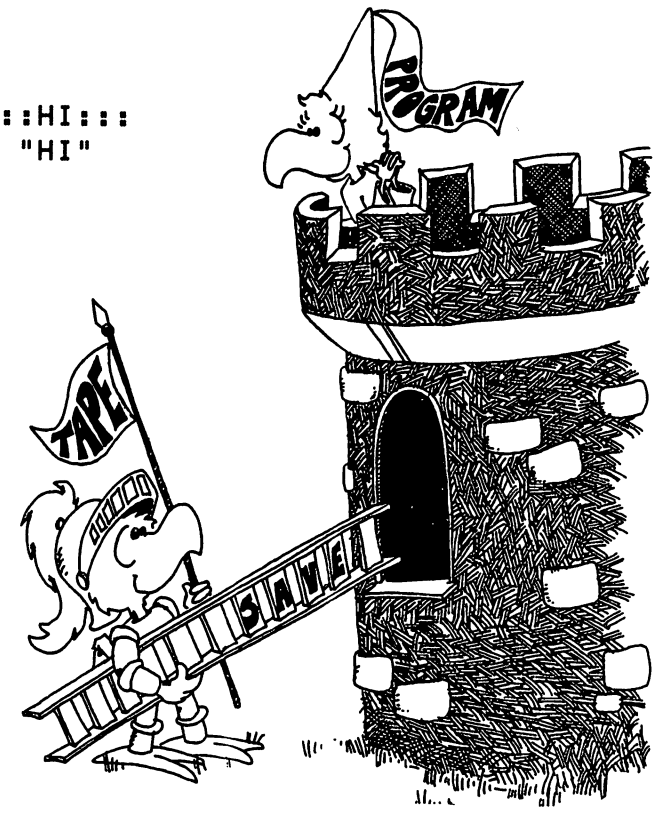
.....

ENTERING A PROGRAM

If you have a program you want to save at this moment, skip to SAVING A PROGRAM.

If not, enter:

```
NEW
10 REM :::HI:::
20 PRINT "HI"
```



SAVING A PROGRAM AND CHECKING THAT IT IS ON TAPE

Turn up the sound on your TV so you can hear the program going onto the tape and the "peeps" that the computer makes.

Put a brand new tape into the recorder and rewind the tape.

If your recorder has a little "speedometer" dial to measure how much tape has been used, press the little button beside the dial. This resets it to zero. (More exactly to "000.")

Enter:

```
SAVE CS1
```

You will hear a peep and see the message:

```
* REWIND CASSETTE TAPE    CS1
  THEN PRESS ENTER
```

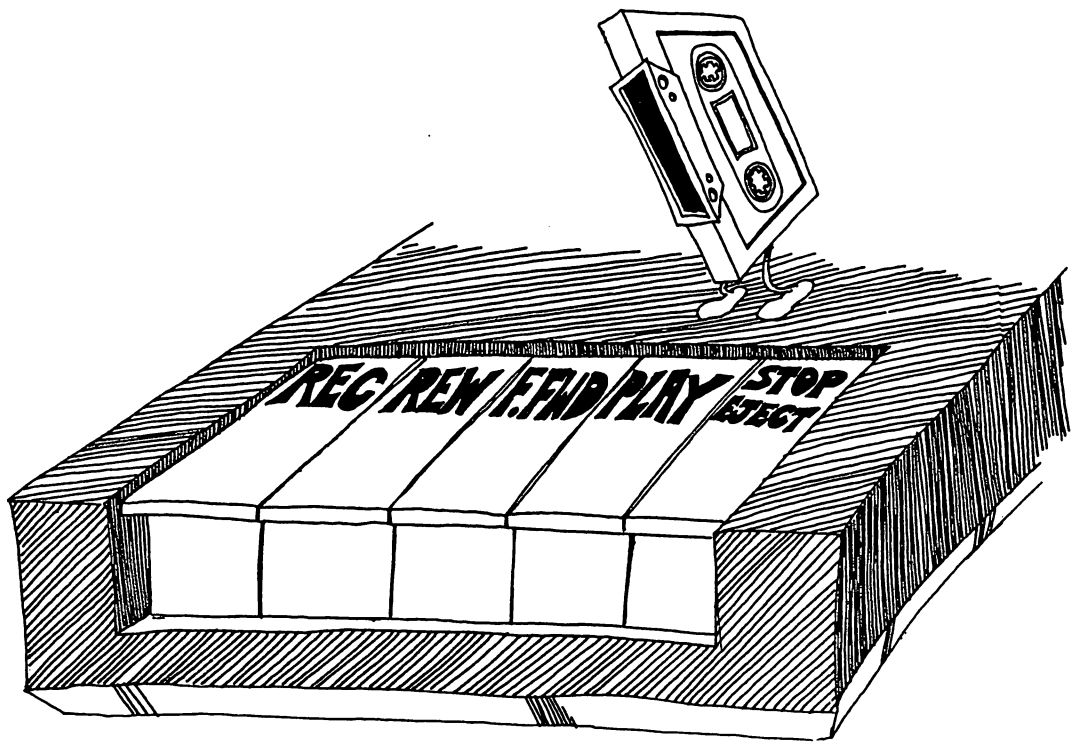
Do it. The computer will peep and say:

```
* PRESS CASSETTE RECORD    CS1
  THEN PRESS ENTER
```

Press the REC and the PLAY keys together, then press the ENTER key.

The computer now peeps and says:

```
* RECORDING
```



There are a few seconds of quiet. Then you hear a steady tone and then a strange twittering like a lovesick gorilla.

The twittering will last a few seconds (for a short program).

Then the computer peeps and prints:

```
* PRESS CASSETTE STOP    CS1
  THEN PRESS ENTER
```

Do it. Now the computer peeps and asks:

```
* CHECK TAPE (Y OR N)?
```

It is a good idea to check to see if the program on the tape is OK.
The computer looks to see if the "checksum" on the end of the recorded program agrees with the sum calculated as the program is read by the computer.

Press the Y key. The computer peeps and prints:

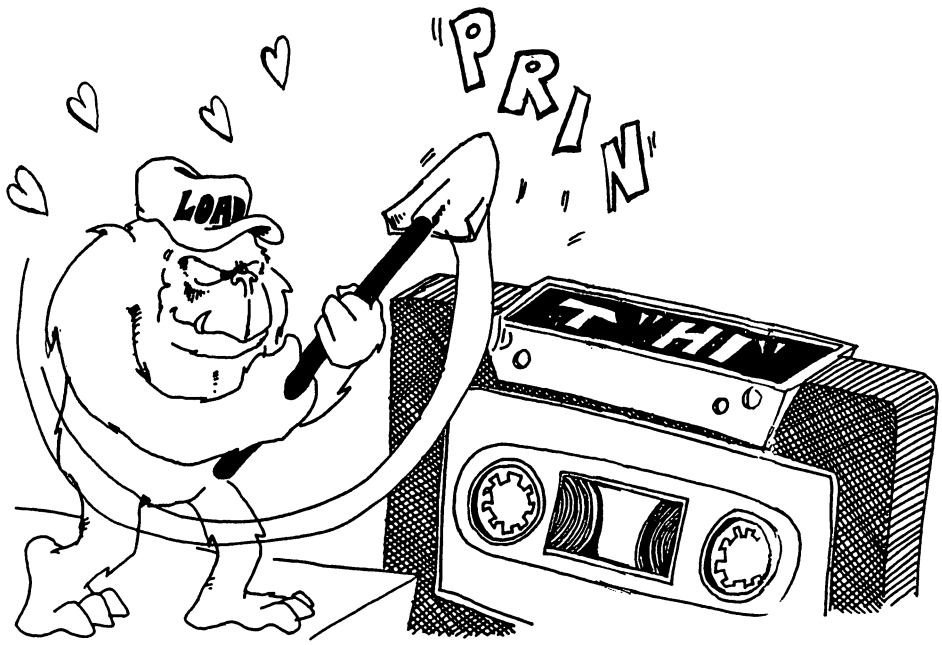
```
* REWIND CASSETTE TAPE    CS1
  THEN PRESS ENTER
```

Do it. The computer peeps and prints:

```
* PRESS CASSETTE PLAY    CS1
  THEN PRESS ENTER
```

Do it. The computer peeps and prints:

```
* CHECKING
```



After a short pause, you should hear the same steady tone and the twittering that you heard when the program was saved.

If everything went well, the computer peeps twice and prints:

```
* DATA OK

* PRESS CASSETTE STOP    CS1
  THEN PRESS ENTER
```

The computer prints the > prompt and the flashing cursor.

BAD LUCK, IT DIDN'T GET SAVED

If the program didn't check out during the

```
CHECKING
```

of your program on tape, the computer will print

either `* ERROR - NO DATA FOUND`

or `* ERROR IN DATA DETECTED`

Then it prints a menu:

```
PRESS R TO RECORD        (again)
PRESS C TO CHECK         (again)
PRESS E TO EXIT          (quit)
```

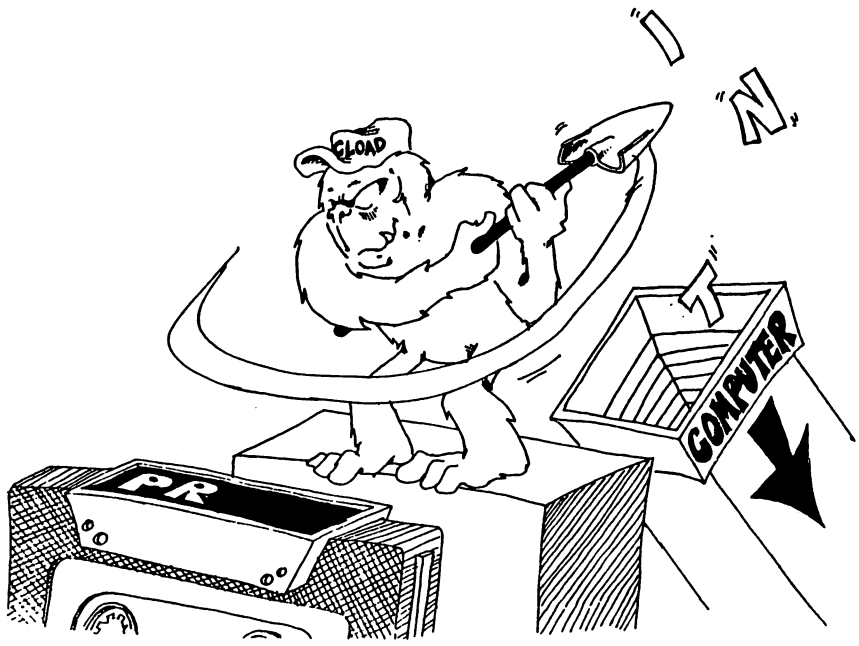
You can check the program again, record it again, or just give up the whole thing.

MAKE A LITTLE LIST

You should write the name of the program on the front of the tape cassette. If your recorder has a little "speedometer" dial, also write the dial number on the front of the tape cassette. This is where the program ends. (It starts at zero or "000" on the dial.)

CAREFUL!

If this is an important program, I suggest you put a second copy on the tape, right after the one you just did. (That is, just start where the directions above say SAVE CS1 but don't rewind the tape when the computer asks you to in the next instruction.)



LOADING THE PROGRAM INTO THE COMPUTER

Let's practice loading the program we just saved.

First, enter NEW

to erase the program from the computer. (Otherwise we won't know if it loaded from tape or just was left over from before.)

Rewind the tape.

Enter: OLD CS1

The computer peeps and prints:

```
* REWIND CASSETTE TAPE    CS1
  THEN PRESS ENTER
```

Do it. The computer peeps and prints:

```
* PRESS CASSETTE PLAY    CS1
  THEN PRESS ENTER
```

the computer peeps and prints:

```
* READING
```

then is quiet for a few seconds. Then you hear the steady tone, then the twittering, then two peeps. The computer should print:

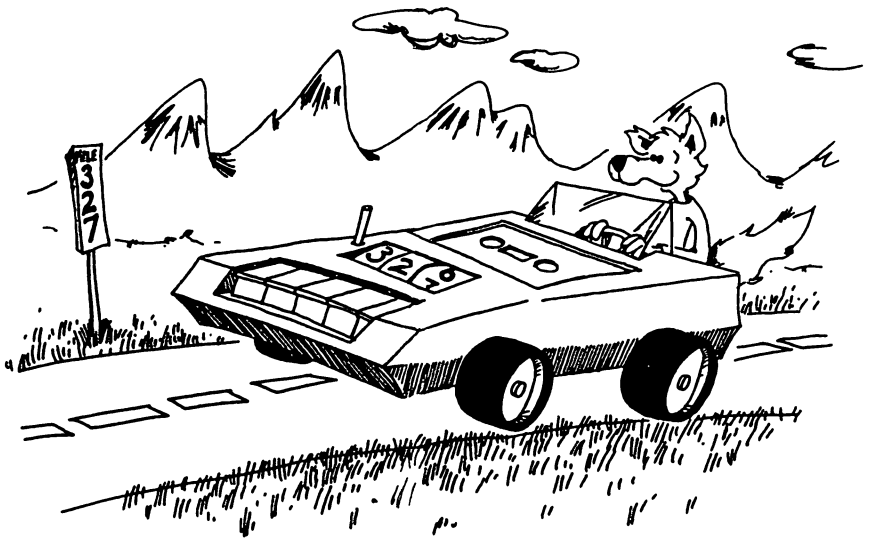
```
* DATA OK

* PRESS CASSETTE STOP    CS1
  THEN PRESS ENTER
```

Now you see the > prompt and the flashing cursor of BASIC.

Enter `LIST`

to see if the program got into the computer memory OK.



HOW MANY PROGRAMS ON ONE TAPE?

You can put several short programs on one tape. If your recorder has a dial, it is easier to find programs on the tape.

But with many programs on a tape, it is harder to find the one you want and more likely that you will make some mistake and ruin a lot of programs.

Assignment 14:

1. Write a short program (4 lines) and SAVE it on tape.
2. Do NEW, and write another short program. SAVE it.
3. Do NEW. Then load and run each program.

INSTRUCTOR NOTES 15 SHORTCUTS AND GRAPHICS

Shortcuts this lesson covers:

LET omission
INPUT with a message in front
LIST in 5 forms

Graphics using the HCHAR and VCHAR statements are introduced.

A brief description of the differences between commands, statements and functions is included in the lesson.

INPUT used without a message in front prints a "?" for a prompt. Used with a message, there is no prompt, and there is no space after the message unless you put one inside the quotes.

We show how to place characters in rows and columns on the screen. The HCHAR command puts single characters or a horizontal line of characters on the screen. Likewise, VCHAR puts a single character or a vertical line on the screen. The character is described by its ASCII number. The numbers for a few punctuation characters useful for graphics are given here. ASCII numbers are described fully in a later lesson.

Learning to use graphics requires mastering several commands and concepts. The students will learn how to choose different colors for her characters, how to move and erase characters giving moving graphics, and how to manufacture custom characters for games or for special fonts.

QUESTIONS:

1. What 5 ways can you use the LIST command?
2. How can you tell that the word LET is missing from a LET command?
3. When do you need a colon in an INPUT statement?
4. What do you put inside the ( ) of a HCHAR( ) statement?
5. Where will the star be put on the screen if your program says:

```
10 CALL VCHAR(10,13,42)
```

LESSON 15 SHORTCUTS AND GRAPHICS

A LET SHORTCUT

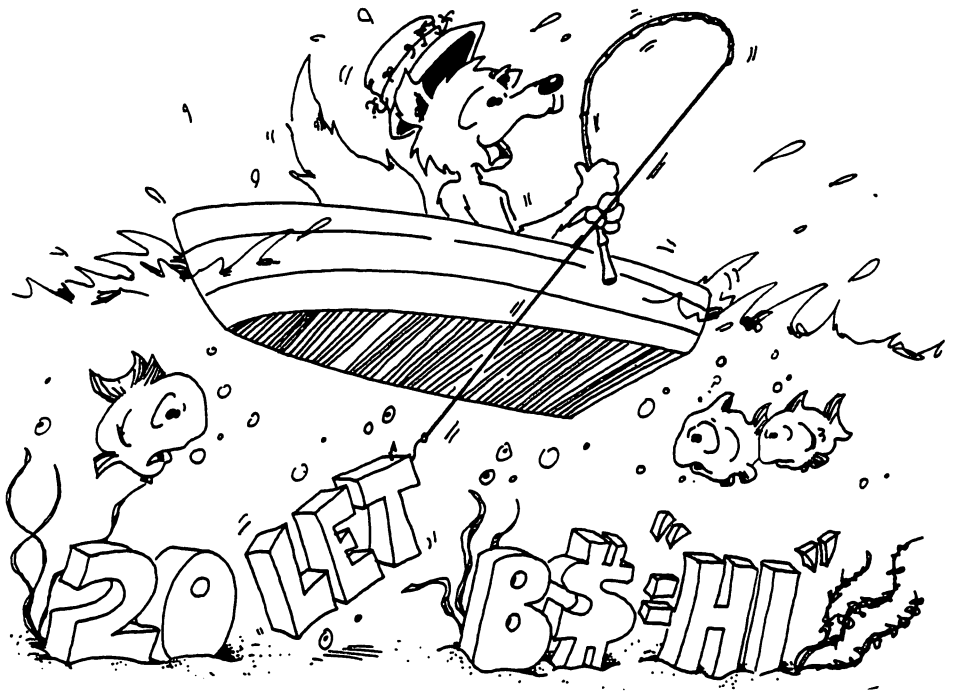
These two lines do the same thing:

```
10 LET A=41        and    10 A=41
```

also these two:

```
20 LET B$="HI"     and    20 B$="HI"
```

You can leave out the word LET from the LET statement! The computer knows that you mean LET whenever the line starts with a variable name followed by an "=" sign.



AN INPUT SHORTCUT

Instead of

```
10 PRINT"ENTER YOUR NAME"
11 INPUT N$
```

You can do:

```
10 INPUT "ENTER YOUR NAME ":N$
```

Put a colon between the message "ENTER YOUR NAME" and the variables.

Examples:

```
10 INPUT "AGAIN? <Y OR N> ": A$
20 INPUT "LOCATION ":X,Y
30 INPUT "MONTH, DAY, YEAR ":M$,D,Y
```

A LIST SHORTCUT

There are 5 ways to use the LIST command:

LIST	lists whole program
LIST 48	lists line 48
LIST 50-75	lists all lines from 50 to 75
LIST -27	lists all lines from beginning to 27
LIST 90-	lists all lines from 90 to the end

KEY WORDS IN TI BASIC

Some key words can be used only as commands. Commands tell the computer to do something. We have learned:

LIST NEW OLD RUN SAVE

```
Right:    LIST
Wrong:    10 LIST
```

Some key words can only be used in statements which are in program lines. We have learned:

STEP GOTO IF THEN INPUT
FOR NEXT

```
Right:    10 INPUT A
Wrong:    INPUT A
```

If you try to use these as commands (without line numbers) the computer prints:

```
          * INCORRECT STATEMENT
or        * CAN'T DO THAT
```

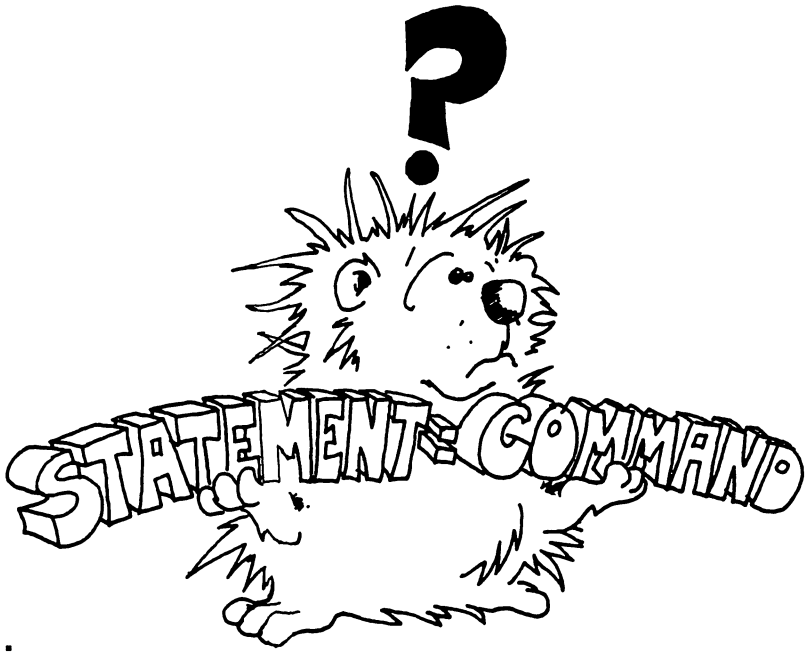
Some key words are functions. We have learned:

INT( ) RND TAB( )

The rest of the key words can be used either as commands or in program lines as statements. We have learned:

CALL XXX LET PRINT REM

```
Right:    10 PRINT
Right:    PRINT
```



Assignment 15A:

1. Write a program which uses each of the shortcuts at least once.
2. Write a "vacation" program. It asks how much you want to spend. Then it tells where you should go or what you should do.
3. Write a "crazy" program which asks your name. The program has three funny ways of saying you are crazy. The program randomly chooses one of these and prints it after your name.

LO-RES GRAPHICS

"Lo-Res graphics" means "low resolution pictures."

It means drawing pictures using dots and lines of dots.

DRAWING DOTS

We will use some punctuation characters to draw pictures.

Each character has a number.

Here are some good ones for pictures:

	32	blank (good for erasing)
*	42	star
+	43	plus
–	45	minus
Ø	48	zero
O	79	letter O

Try adding more HCHAR statements to draw more dots and other punctuation marks on the screen.

Look: `30 CALL HCHAR` (row, column, character number)

You can put any number from 1 to 24 in the row place.

You can put any number from 1 to 32 in the column place.

You can use any character number from 32 to 126.

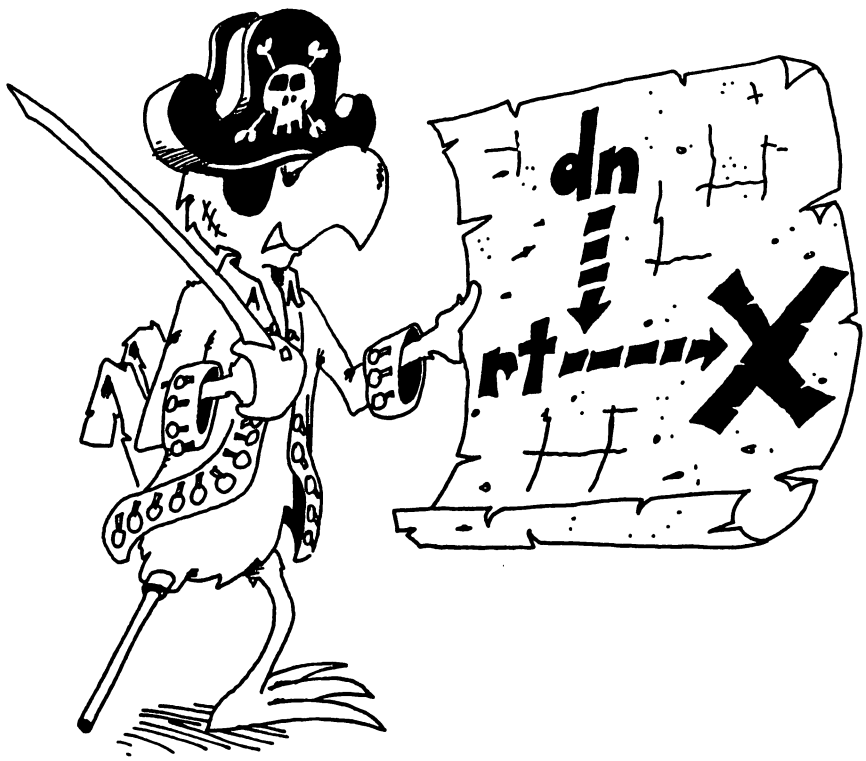
Try this:

```
10 REM ONE STAR
20 CALL CLEAR
30 CALL HCHAR(3,7,42)
99 GOTO 99
```

Press FCTN CLEAR to end the program.

Line 30 does this:

goes down 3 rows from the top
counts across 7 spaces from the left
puts character 42 there
(character 42 is the "star")



Try this:

```
10 REM CHARACTERS
20 CALL CLEAR
25 FOR C=33 TO 126
30 CALL SOUND(100,900,10)
32 X=INT(RND*32)+1
34 Y=INT(RND*24)+1
40 CALL HCHAR(Y,X,C)
50 FOR T=1 TO 200
51 NEXT T
90 NEXT C
```

DRAWING HORIZONTAL LINES

Use CALL HCHAR to draw horizontal lines:

Try this:

```
10 REM HORIZONTAL LINES
15 CALL CLEAR
20 CALL HCHAR(5,3,43,20)
```

It is just the same as drawing a dot, but you have one more number inside the ( ).

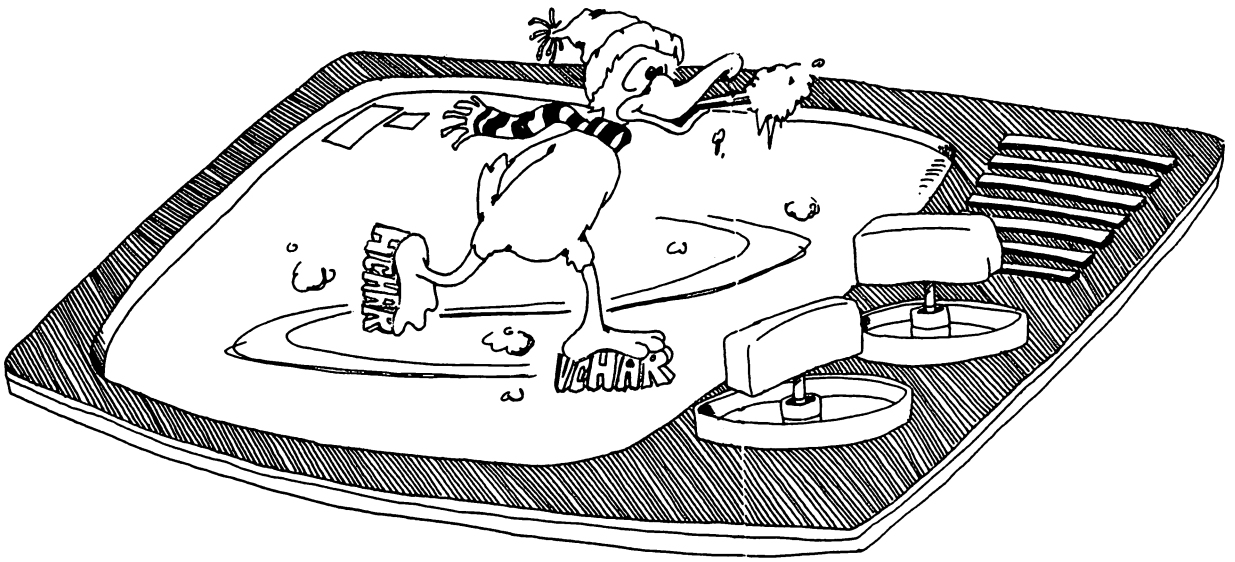
```
2  CALL HCHAR (row, column, character, how many)
```

So:

```
20 CALL HCHAR(3,7,88,8)
```

Means: count 3 rows down from the screen top
then go 7 across
put character 88 (the "X" letter) there
then put 8 more characters across to the right.

The "H" in HCHAR means "horizontal" line.



DRAWING VERTICAL LINES

Add: `70 CALL VCHAR(8,4,33,7)`

Means: count 8 rows down
go 4 characters across
put the character 33 there
then put 6 more down from there

Assignment 15B:

1. Put a dot at 11 down and 7 across.
2. Put a horizontal line of stars on line 3 down and running from 4 to 12.
3. Put a vertical line of dots starting 4 down and 6 across. Make it 9 characters long.

INSTRUCTOR NOTES 16 THE IF STATEMENT WITH NUMBERS, END

The IF command is extended to numerical expressions. The END statement is introduced. The logical relations used in this lesson are:

=, >, <, <>

The > and < symbols may still confuse your student. Remind her that "the large end of the > goes beside the large number."

The IF statement is intricate because it involves two elements: a test for truth and a possible branch.

Because of the branch, the IF may be part of a non-compact section of code in the program. Most of the confusion comes in seeing what lines in the program logically go together, and how the flow of control moves through the program.

The easiest case is where the IF comes at the end of a section and the branch is back up to the beginning of the section. This is a DO UNTIL... and the exit occurs when the IF fails the test.

Another common case is the "skip or fall through." When the IF is TRUE, some lines are skipped and the next section of code is executed. Otherwise the lines are executed and control "falls through" to the next section of code.

An elaboration of this avoids the "fall through" by having a GOTO at the end of the section. This forms a DO WHILE... .

The use of nested IF's is demonstrated.

A "home made" loop is demonstrated in the GUESSING GAME but not discussed. The loop starts in line 50 and goes to 81. The exit test is made in line 61.

QUESTIONS:

1. What part of the IF command can be TRUE or FALSE?
2. What follows the THEN in an IF command?
3. After this little program runs, what will be in box D?

```
10 LET D=4
15 IF 3 < 7 THEN 30
20 LET D=9
30 REM
```

4. Same question, but for 3 > 7.

LESSON 16 THE IF STATEMENT WITH NUMBERS, END

Try this:

```
10 REM *** TEENAGER ***
15 CALL CLEAR
20 PRINT"YOUR AGE?"
30 INPUT A
40 IF A<13 THEN 60
50 IF A>19 THEN 70
55 PRINT"YOU ARE A TEENAGER!"
56 END
60 PRINT"NOT YET A TEENAGER!"
61 END
70 PRINT"GROWN UP ALREADY!"
90 END
```

This IF command is like the one that you used before with strings. Again we have:

`10 IF` phrase A is true `THEN` line number

"Phrase A" can have these arithmetic symbols:

=	equal to
>	greater than
<	less than
<>	not equal to

Each "phrase A" is written in "math language" but you should say it out loud in English. For example:

A <> B	is pronounced	"A is not equal to B"
5 < 7	is pronounced	"five is less than seven"

THE END COMMAND

The program may have zero, one, or many END commands.

Rule: The END command tells the computer to stop running and go back to the command mode.

That is really all it does. You can put an END command anywhere in the program.

PRACTICE

For these problems, LET A=7 and LET B=5 and LET C=5.

Say each "phrase A" out loud and circle T or F for true or false:

A=B T F
A>B T F
A<B T F
A<C T F
B=C T F
B<C T F
A<>B T F
B<>C T F

Example, say: "A is equal to B, or 7 is equal to 5, that is FALSE".

or "A is greater than B, or 7 is greater than 5, that is TRUE."

GUESSING GAME

```
10 REM --- GUESSING GAME ---
15 CALL CLEAR
20 REM----------- INSTRUCTIONS
21 PRINT "TWO PLAYER GAME"
25 PRINT
30 PRINT "FIRST PLAYER ENTER A NUMBER FROM 1 TO 100"
35 PRINT "WHILE SECOND PLAYER ISN'T LOOKING"
40 REM------------CHOOSE NUMBER
41 INPUT N
45 CALL CLEAR
47 PRINT
50 REM------------MAKE GUESSES
51 PRINT "MAKE A GUESS"
52 PRINT
55 INPUT G
60 REM------------IS THE GUESS RIGHT?
61 IF G=N THEN 90
65 REM------------IS THE GUESS TOO SMALL?
66 IF G<N THEN 70
67 REM------------GUESS WAS TOO LARGE
68 PRINT "TOO LARGE"
69 GOTO 80
70 REM------------GUESS WAS TOO SMALL
71 PRINT "TOO SMALL"
80 REM -----------GET ANOTHER GUESS
81 GOTO 50
90 REM----------- THE GAME IS OVER
92 PRINT
95 PRINT "THAT'S IT!"
```

Save on tape.

Line 81 usually sends you to line 50 so that more guesses can be made. But if the guess was right in line 61, then you skip to line 90 and the program prints the message:

```
"THAT'S IT!"
```

Assignment 16:

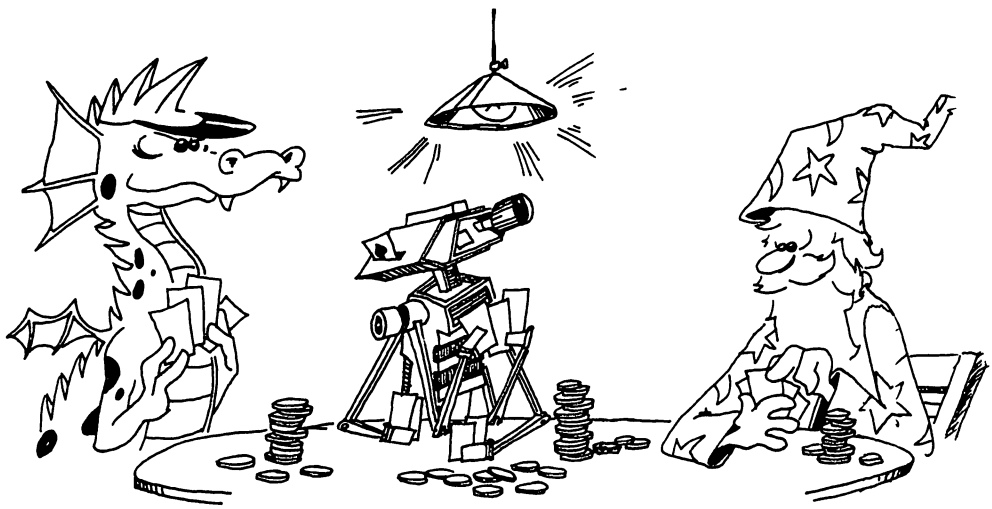
1. Draw the road map for this program. Lines 61 and 66 have "forks in the road." Lines 69 and 81 are jumps to other lines.
2. Here is another program. What will it print, and how many times?

```
10 N=1
20 IF N <> 13 THEN 30
25 PRINT "UNLUCKY"
30 LET N=N+2
40 IF N>30 THEN 99
50 GOTO 20
99 PRINT "DONE"
```

   What will it print if line 10 is changed to:

```
10 LET N=2
```

3. Write a program which says something about each number from one to ten. The player enters a number and the computer prints something about each number: "three strikes, you're out" or "seven is lucky" etc.
4. Write a digital clock program. It uses a timing loop to count seconds. Input the present time in hours, minutes and seconds. The clock then counts seconds and prints them out. When 60 seconds have gone by, add one to the minutes and put seconds back to zero. Same with hours. Run the clock a long time and adjust the timing loop so the clock keeps good time.
5. Write a game for guessing a card that someone has entered. You must enter the suit (club, diamond, heart, or spade) and the value (1 through 13). First they guess the suit, then the program goes on to ask the value. Keep score.



INSTRUCTOR NOTES 17 COLOR GRAPHICS

The CALL COLOR( ) command in TI BASIC lets you put up to 15 colors on the screen at once.

CALL COLOR (char. set, foreground, background)

is a command that colors a whole set of 8 characters at once. There are 16 sets. The characters are numbered from 32 to 159. Set 1 is from 32 to 39, set 2 the next 8 characters, etc. Normally, the characters are the ASCII set listed in the appendix.

Because you assign two colors to each character set, a foreground color and a background color, you have an immense number of color combinations.

You can choose a "transparent" color which lets the background color, assigned by a CALL SCREEN( ) command, show through in either the character foreground or background. (If transparent is used for the foreground, you have a "reversed" character.)

The background normally only shows on the border because the "field," where the characters are printed, usually starts out filled with character 32, the blank, which belongs to set 1.

The CALL COLOR command recolors all the characters in the set: those yet to be placed on the screen and THOSE ALREADY ON THE SCREEN!

Coloring in sets and changing those already on the screen may sound dreadful but, properly used, you can draw almost anything you wish.

You assign a different color pair to each of 16 character sets at the beginning of the program. Then you redefine characters in each set until you can make the pictures you want in the colors you want. You may want to reassign colors or characters in the body of the program to obtain special effects, but this may complicate the program somewhat.

QUESTIONS:

1. What character set does the "star" belong to? The "space"?
2. How many colors can you put on the screen at once?
3. What does "character color" and "background color" mean in the CALL COLOR command?
4. How many characters are in each character set?
5. Which character numbers belong to set 2?

LESSON 17 COLOR GRAPHICS

The TI 99/4A can put 15 colors on the screen at once.

THE COLORS ARE NUMBERED

1 TRANSPARENT	9 MEDIUM RED
2 BLACK	10 LIGHT RED
3 MEDIUM GREEN	11 DARK YELLOW
4 LIGHT GREEN	12 LIGHT YELLOW
5 DARK BLUE	13 DARK GREEN
6 LIGHT BLUE	14 MAGENTA (PURPLE)
7 DARK RED	15 GREY
8 CYAN (BLUE- GREEN)	16 WHITE

We will explain color number 1, transparent, in a later lesson.

ADJUSTING YOUR TV SET

When you first turn on the computer (or when you press FCTN QUIT) you see two rows of colored rectangles on the screen.

Adjust the TV controls to make the colors look right.

You should see all the colors in the list above, especially a good "yellow."

PICKING THE BORDER COLOR

Pick a color for the background of the whole screen.

Run:

```
10 REM COLORS
15 CALL CLEAR
20 CALL SCREEN(15)
99 GOTO 99
```

We picked color 15, which is grey.

Press FCTN CLEAR to end the program.

CHARACTER SETS

Here are the first two character sets:

Set 1		Set 2	
32	space	40	(
33	!	41	)
34	"	42	*
35	#	43	+
36	$	44	,
37	%	45	-
38	&	46	.
39	'	47	/

The rest of the character sets are listed in the TI USER'S REFERENCE MANUAL.

Each character has a number, called the ASCII number.

(ASCII is pronounced "ask-key".)

Important! When you choose colors, all the characters in a set are given the same color.

PAINTING CHARACTER SET 2

The command is:

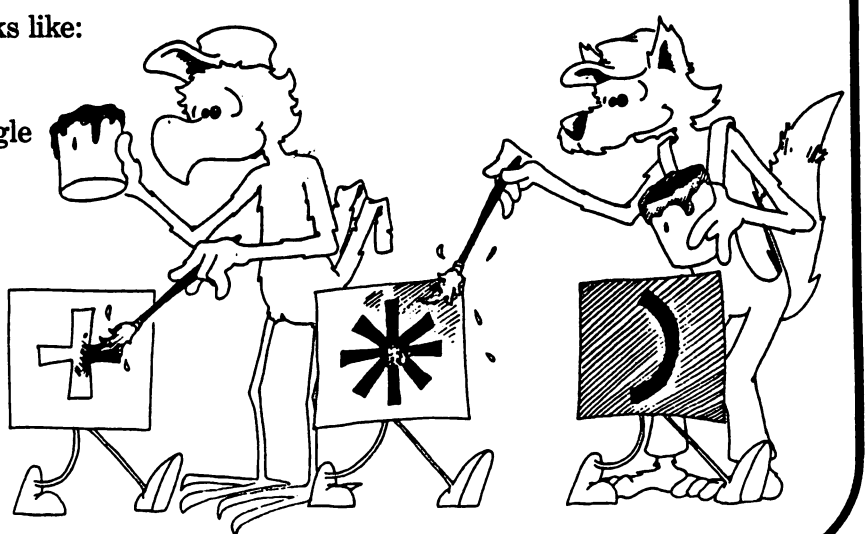
`CALL COLOR` (set, character color, background color)

Add: `30 CALL COLOR(2,9,2)`

Means: `30 CALL COLOR` (set 2, red characters, black background)

So each character of set 2 looks like:

a red character
on a little black rectangle



PUT THEM ON THE SCREEN

Use the commands CALL HCHAR and CALL VCHAR:

Add:

```
40 REM PRINT SETS 1 AND 2
41 FOR CH = 32 TO 47
45 CALL HCHAR(10,CH-31,CH)
50 FOR T=1 TO 300
51 NEXT T
60 NEXT CH
```

And run it. You see all the characters of set 1 and set 2. The characters of set 2 are red inside black rectangles.

Assignment 17A:

1. Write the name of your favorite movie and movie star and then make them change colors on the screen.
2. Now draw a "theater marquee" around the names and have it flash a different color.

MOVING PICTURES

Run:

```
10 REM HUMPTY DUMPTY IS SQUARE
15 CALL CLEAR
20 CALL SCREEN(2)
25 REM-------------- MAKE A SQUARE CHARACTER
26 CALL CHAR(42,"FFFFFFFFFFFFFFFF")
27 REM 16 LETTER F's
30 FOR C=3 TO 16
35 CALL COLOR(2,C,2)
40 FOR J=1 TO 23
50 REM-------------- ERASE THE OLD SQUARE
51 CALL HCHAR(J,C*2-2,32)
55 REM-------------- DRAW THE NEW SQUARE BELOW
56 CALL HCHAR(J+1,C*C-2,42)
60 NEXT J
70 NEXT C
```

Notice these things:

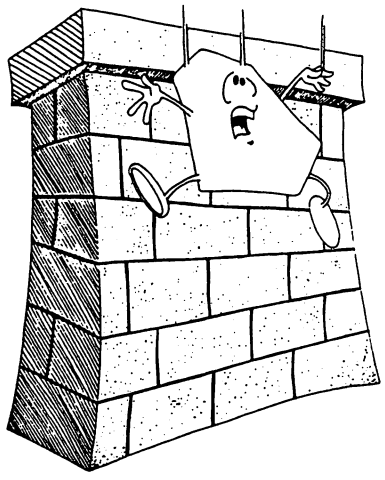
Line 26 makes a new character, number 42. It is a rectangle. The old 42 was a star. The command CALL CHAR( ) makes the new 42.

You will learn how to make any character you want in a later lesson.

Line 35 keeps changing the color of characters in set 2. When the color changes, ALL the characters in that set change color, EVEN THE ONES THAT WERE PUT ON THE SCREEN EARLIER!

Line 50 erases the old square before line 55 draws the new one. It looks like the square is falling.

When the program ends, character 42 is changed back into a star.



THE SPACE CHARACTER IS IN SET 1

Run:

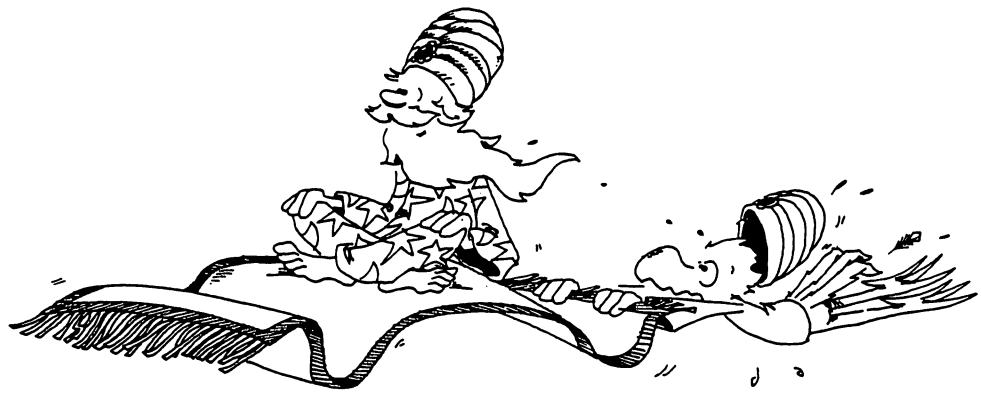
```
10 REM THE CHARACTER FIELD
15 CALL CLEAR
16 PRINT "! #$%↑&*()+"
20 CALL SCREEN(14)
30 FOR C=3 TO 16
35 CALL COLOR(1,2,C)
36 PRINT C
40 FOR T=1 TO 500
41 NEXT T
50 NEXT C
```

Character 32, which is a "space," is in set 1.

Most of the screen is usually "filled" with this blank character.
So most of the screen changes color if you change the color of set 1.

Assignment 17B:

1. Change the HUMPTY DUMPTY program so that the square moves across the screen instead of down.
2. Add to the number guessing game in lesson 13 so that a colored picture shows when the correct answer is guessed. Use a timing loop so that the picture shows for a few seconds before the game starts again.
3. Write a program to draw "Sinbad's Magic Carpet," a rectangular pattern of colored dots on the screen.



INSTRUCTOR NOTES 18 COMMAND AND RUN MODES

This lesson explains the Command Mode and the Run Mode of the computer.

We placed this material rather late in the book, despite its fundamental nature, because it is abstract and because we did not wish to slow down the race to mastery of the core commands in BASIC.

However, you may want to take up this chapter at some earlier time in the course. The only commands used in this lesson are:

NEW, PRINT and RUN

Other names for these modes are:

Command Mode:	direct mode	immediate mode
Run Mode:	deferred mode	

In some computers, the command mode is called "the edit mode" but TI BASIC has a line edit mode called "the edit mode."

The command mode is the home base of the computer user. In the command mode you enter a line. The characters go into the input buffer.

When RETURN is pressed, the computer looks to see if the line starts with a number. If so, it stores the line in the program space, making room at the right location so that the lines are numbered in order.

A line must start with a line number followed by a "statement" key word. Or it must start with no number and a "command" key word. Otherwise, an error message will be printed.

QUESTIONS:

1. What does the computer do in the "RUN mode"?
2. How can you tell if the computer is in the "command mode"?
3. What 3 kinds of things can you do in the command mode?
4. If you enter a line which starts with a line number, what happens to the line?

LESSON 18 COMMAND AND RUN MODES

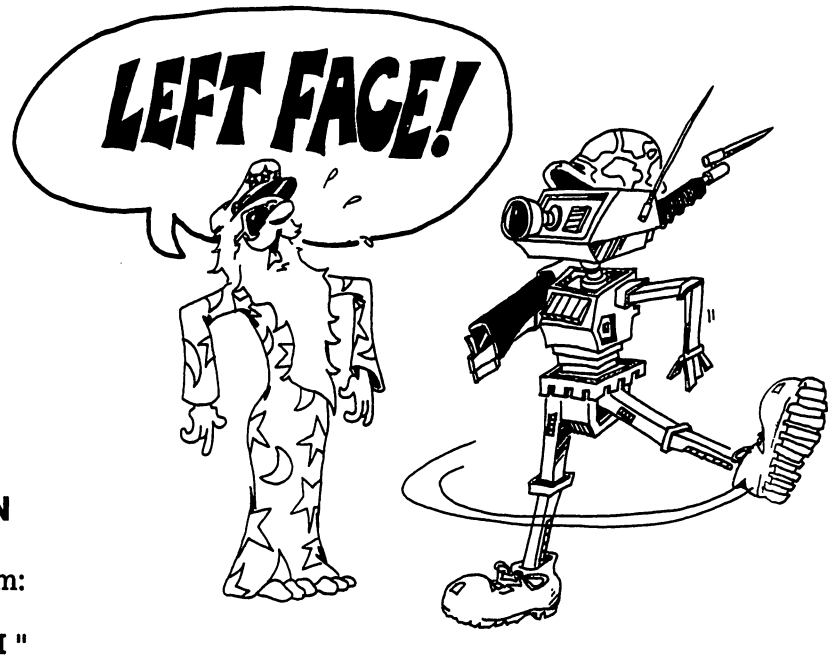
Enter: NEW

You are ready to begin the lesson.

EXECUTION AND RUNNING

We mean "execution" like the soldier executing the command "Left Face," not "execution by firing squad."

"Execute a program" means the same as "run a program."



PROGRAM EXECUTION

Enter and run this program:

```
10 PRINT "HI"
```

This is the usual way to make and run programs. You enter program lines.

Each line starts with a number and followed by a statement. The computer stores the line with the other lines in memory. Later you execute the program by entering the command "RUN."

COMMAND EXECUTION

Here is a short cut. Enter this (no line number in front):

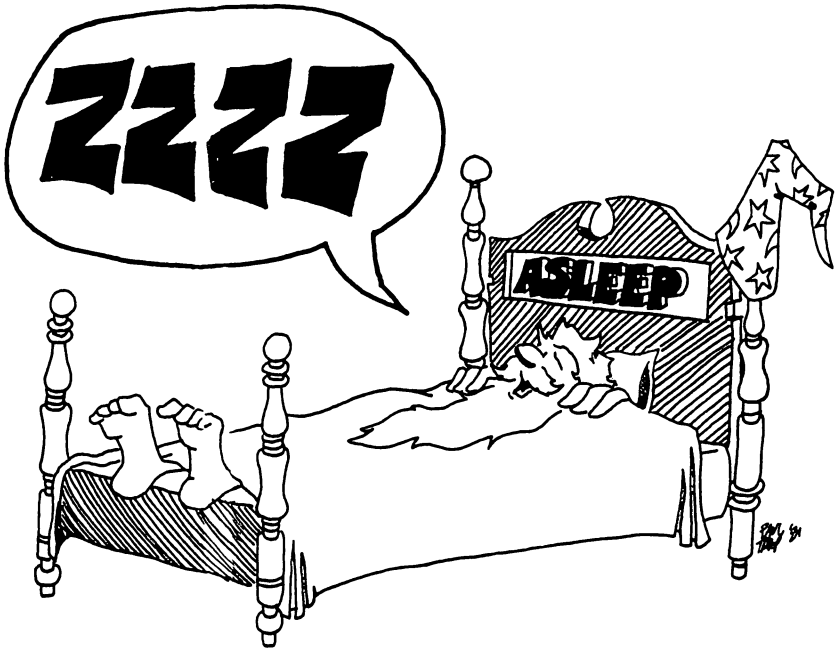
```
PRINT "HI"
```

This time PRINT is used as a command. The computer executes the command right away, without waiting for you to enter RUN.

ASLEEP OR AWAKE?

People act one way if they are awake and another way if they are asleep. They have two "operating modes."

You can tell if they are asleep because they snore. (Well, not all people snore, but to explain how computers are like people, let's pretend that all sleeping people snore.) The computer has two operating modes too. They are called the "command mode" and the "RUN mode."



THE COMMAND MODE

Enter: NEW

You see the ">" symbol and the flashing cursor square.

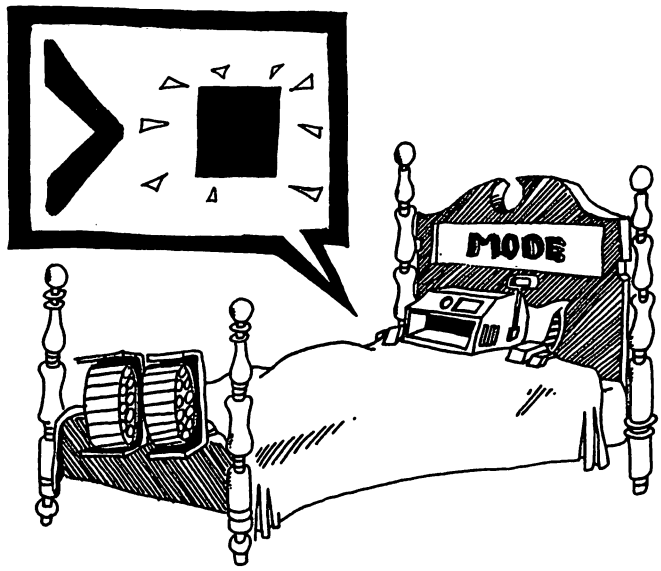
The ">" is called a "prompt" and says that the computer is in the "command mode" of TI BASIC.

The ">" is the "snoring" of the computer when it is in the command mode.

In the command mode the computer just waits for you to enter something.

While the computer is in the command mode:

You can enter programs by typing lines which start with numbers.
You can use the computer like a pocket calculator. Big pocket!
You can enter commands like LIST, CALL CLEAR, PRINT, etc.



THE CALCULATOR

You can do arithmetic in the Command Mode. Try this:

```
PRINT 3+7
```

The computer prints the answer "10".

THE RUN MODE

Enter RUN to leave the command mode and go to the run mode.

While the computer is in the run mode:

The screen turns green.

The program in memory runs.

When the program is finished, the computer automatically goes back to the command mode.

Assignment 18:

1. How can you tell if the computer is in the command mode?
2. How can you tell if the computer is in the RUN mode?
3. What mode does the computer enter when the program is done running?
4. How can you tell where the next letter you type will appear on the screen?

INSTRUCTOR NOTES 19 DATA, READ, AND RESTORE

This lesson concerns the DATA statement. You put data into the DATA statement at the time you write the program. READ gets data from the DATA statements and RESTORE puts the pointer back to the beginning of a DATA statement.

The storing of data in DATA statements has a few confusing aspects when first confronted. You can never change any of the data in the statement unless you rewrite the program. Of course, you can READ the data into a variable box, then change what's in the box.

You must READ the data to be able to use it. It must be read in order, starting from the beginning. If you want to skip some data that is in a given DATA statement, you have to read and throw away the stuff before it. (This procedure is not discussed in the lesson; it may be mentioned to the student when other ideas about DATA are well entrenched.)

In TI BASIC you can skip data by arranging it in different DATA statements, and pointing to the one you want with a RESTORE nnn statement, where nnn is the number of the DATA statement.

The idea of a "pointer" is used in this lesson. A pencil in the hand of the instructor, pointing to items in a DATA statement, helps clarify this concept.

Using DATA saves some error prone typing if you have a lot of data.

However, it is also useful in cases where there is not really very much data because it clearly separates the actual data from the processing of the data. This helps when debugging programs.

One of the most common uses of DATA is to fill arrays with initial values.

QUESTIONS:

1. What happens if you try to READ more data items than are in the DATA statements?
2. What rule tells you where to put the DATA statements into the program? How about where to put the READ statements?
3. Can you put numerical data and string data into the same DATA statement?
4. Can you change the items in a DATA statement while the program runs?

LESSON 19 DATA, READ, AND RESTORE

TWO KINDS OF DATA

There are two kinds of data in your programs:

1. The data you INPUT or get by CALL KEY through the keyboard.

```
10 REM FIRST KIND OF DATA
20 CALL CLEAR
30 PRINT"YOUR PET PEEVE"
35 INPUT P$
37 CALL CLEAR
40 PRINT"REALLY!"
50 PRINT"YOU DON'T LIKE ";
60 PRINT P$;"?"
```

   In this program P$ is data entered by the user as the program runs.

2. The data which is stored in the program at the time it is written.

```
10 REM THE SECOND KIND OF DATA
20 CALL CLEAR
30 X=2
40 Y=3
50 PRINT X+Y
```

   In this program X and Y are data stored in the program by the programmer when she wrote the program.

STORING LOTS OF DATA

It is OK to store small amounts of data in LET statements. But it is awkward to store large amounts of data that way.

Use the DATA statement to store large amounts of data.

Use the READ statement to get the data from the DATA statement.

```
10 REM LOTS OF DATA
20 CALL CLEAR
30 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY
FRIDAY,SATURDAY
40 READ D1$,D2$,D3$,D4$
60 PRINT D1$,D2$
```

After the program runs, box D1$ holds the first item in the DATA list (SUNDAY) and box D2$ holds the second (MONDAY), etc.

STRANGE RULES

1. It doesn't matter where the DATA statement is in the program.

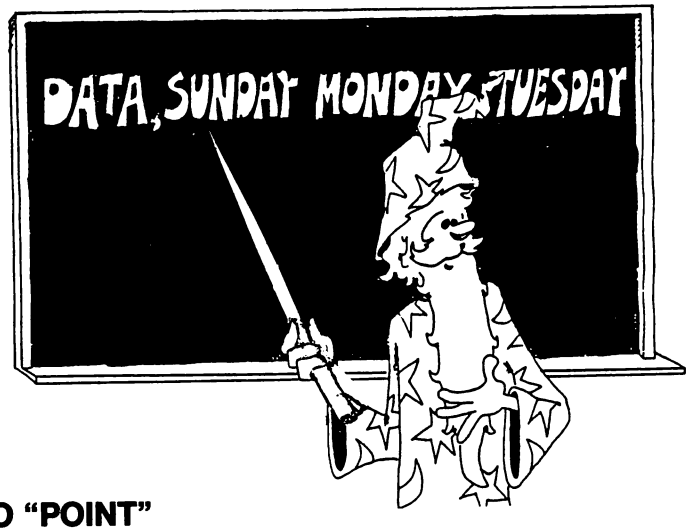
Do this: Change line number 30 in the above program to line number 90. Run the program. It works just the same.

2. It doesn't matter how many DATA statements there are.

Do this: Break the DATA statement into two:

```
90 DATA SUNDAY,MONDAY,TUESDAY
91 DATA WEDNESDAY,THURSDAY,FRIDAY,SATURDAY
```

Run the program. It works just the same as before.



IT IS POLITE TO "POINT"

READ uses a pointer. It always points to the next item to be read.

You can't see the pointer. Just imagine it is there.

When the program starts, the READ pointer points to the first item in the first DATA statement in the program. (That is, the DATA statement with the lowest line number of all DATA statements in the program.)

Each time the program executes a READ command, the pointer moves to the next item in the DATA list.

If the pointer gets to the end of one DATA statement, it automatically goes to the next DATA statement. (That is, to the DATA statement with the next higher line number.)

It doesn't matter if there are a lot of lines between.

Do this: Change line 90 back to line 30. (Leave line 91 alone.)

```
30 DATA SUNDAY, MONDAY, TUESDAY
        . . .
91 DATA WEDNESDAY,THURDSAY,FRIDAY,SATURDAY
```

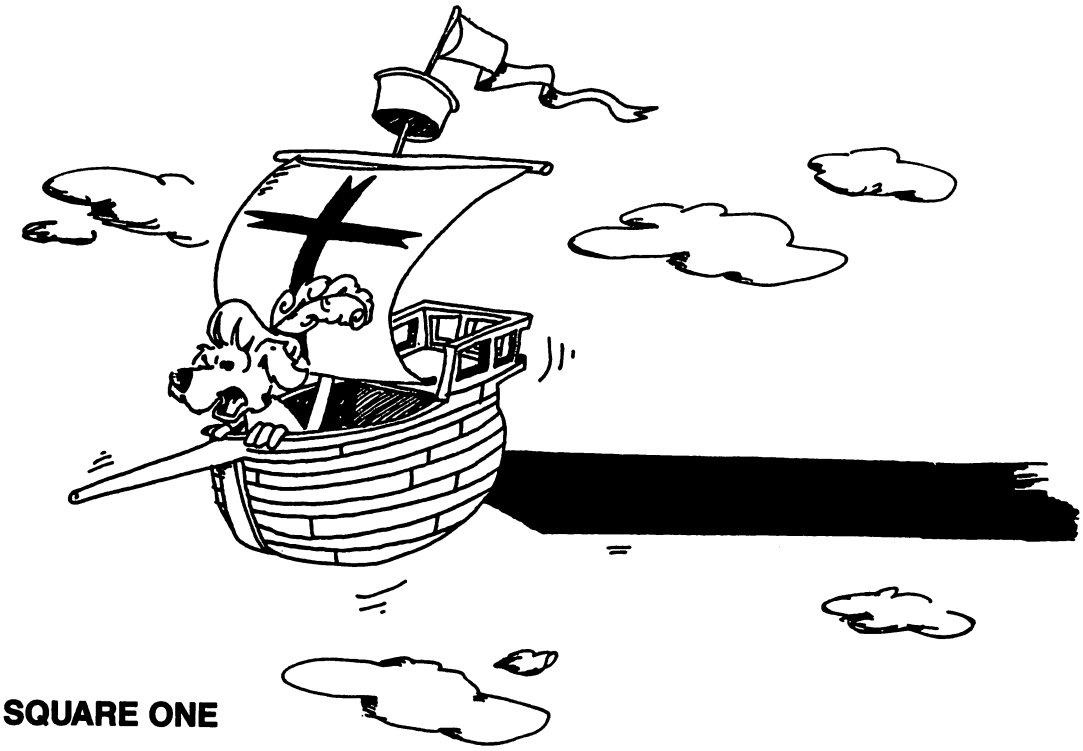
Run the program. It works just the same.

FALLING OFF THE END OF THE DATA PLANKS

When the pointer reaches the last item in the last DATA statement in the program, there are no more items left to read. If you try to READ again, you will see an error message:

`DATA ERROR` or

`DATA ERROR IN nnn` where nnn is a DATA line number



BACK TO SQUARE ONE

At any point in the program, you have only three choices for the READ pointer.

1. You can do another READ: Then the pointer moves ahead one item.
2. You can command RESTORE: Then the READ pointer is put back to the beginning of the first DATA statement in the program.
3. You can command RESTORE nnn: The nnn is a line number of a DATA statement. The READ pointer is put on the first item in that DATA statement.

MIXTURES OF DATA

The DATA statement can hold strings or numbers in any order.

But you must be careful in your READ command to have the correct kind of variable to match the kind of data.

Correct:

```
70 DATA 77,FUZZ
75 READ N
80 READ B$
```

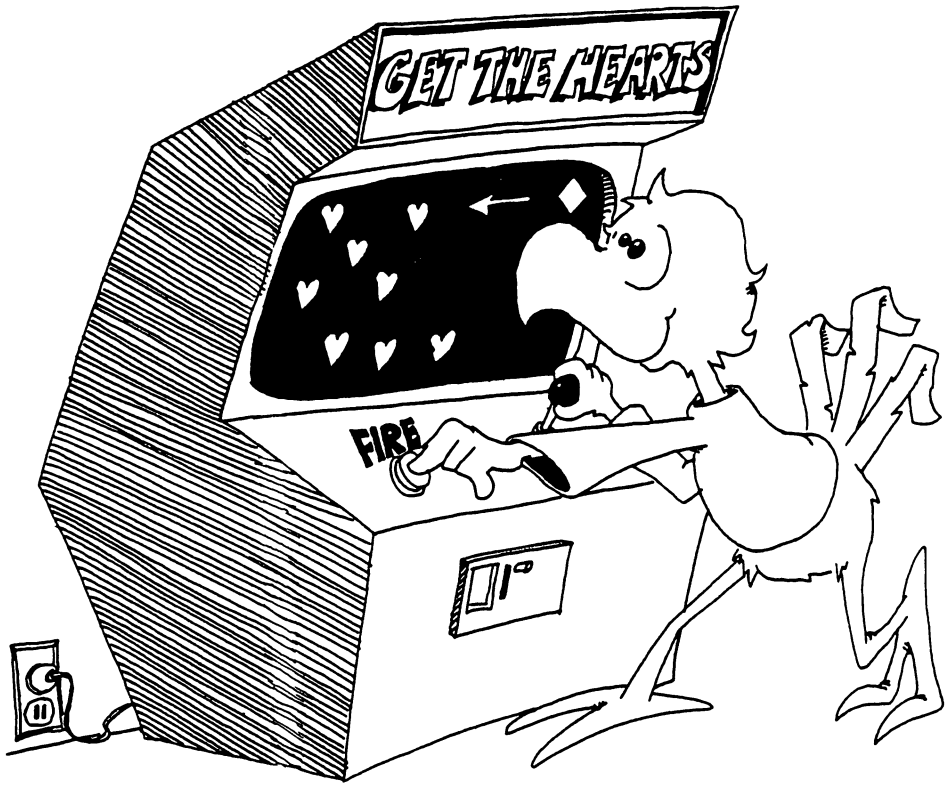
Wrong:

```
70 DATA 77,FUZZ
75 READ B$          OK, B$ box holds "77"
80 READ N           TYPE MISMATCH ERROR
```

You can't put "FUZZ" into a number box.

Assignment 19:

1. Write a program naming your relatives. When you ask the computer "UNCLE" it gives the names of all your uncles. DATA statements will have pairs of items. The first item is a relation like FATHER or COUSIN. The second item is a person's name. Of course, you may have several brothers, for example, each with a DATA statement.



INSTRUCTOR NOTES 20 SOUND

The CALL SOUND command turns on from 1 to 4 voices. You can set the pitch and loudness of each voice and the duration (up to 4.25 seconds) of the whole combination.

The computer continues to execute instructions while the sound is being made. This is a very convenient feature of the TI 99/4A computer.

You have a choice of whether a new CALL SOUND command will cut off the existing sound or wait for it to end. If it waits, the execution of other program commands must wait too.

The number specifying pitch is actually the frequency in Hertz or cycles per second. Frequencies from 110 Hz. (a low bass note) to tones above human perception are allowed, in one Hz. steps. The middle A for tuning an orchestra is 440 Hz.

A variety of noise sounds can be made by one voice, while up to three "musical" tones are also sounding.

The most interesting sounds will be made using several SOUND comands one after another to provide variations such as attack, sustain, and decay portions of a sound.

When using sound in graphics situations, you get the most elaborate effects if you interweave the sound commands with the "move the graphics" commands.

The DATA command is useful for storing the notes in music.

QUESTIONS:

1. Which pitch numbers give deep sounds? Which give high notes?
2. What turns the sound off?
3. What is the longest time the sound will last?
4. How do you make a hissing noise?
5. What number gives the loudest noise? The softest?
6. How do you make a "motor" sound?

LESSON 20 SOUND

The TI-99/4A has four sound voices.

One voice can "sing" by itself.

Or two, three, or four can "sing" at the same time.

You can pick one of the voices to be a "noise maker," with a choice of 8 kinds of noises.

Use them in music and sound effects like explosions, laser guns, and wind.

MAKING MUSIC

Each CALL SOUND command picks one sound duration and 1, 2, 3, or 4 voices. Each voice has a pair of numbers giving its pitch and loudness.

Examples:

```
30 CALL SOUND(D, P , L )
30 CALL SOUND(D, P1, L1, P2, L2)
30 CALL SOUND(D, P1, L1, P2, L2, P3, L3)
30 CALL SOUND(D, P1, L1, P2, L2, P3, L3, P4, L4)
```

variable	value
D for "duration"	from 1 to 4250
P for "pitch"	from 110 to 44733
L for "loudness"	from 0 to 30

DURATION

"Duration" means "how long the sound lasts."

The number D varies from 1 to 4250. Move the decimal over 3 places and D tells how many seconds the sound will last.

D = 500	half a second
D = 1000	1 second
D = 1500	1.5 seconds
D = 2000	2 seconds
D = ...(fill in)	3 seconds
D = ...(fill in)	4 seconds

PITCH

Pitch tells whether you have a "high note" or "low note." The bigger the number, the higher the pitch.

The pitch numbers are actually the frequency in Hertz or cycles per second. Only integers work for pitch numbers.

Small numbers give low tones. Large numbers give high notes, maybe so high only your dog can hear them!

Here is a tempered scale of musical notes:

note	number
C (below middle C)	131
C #	139
D	147
D #	156
E	165
F	175
F #	185
G	196
G #	208
A	220
A #	233
B	247
C (middle C)	262
C #	277
D	294
D #	311
E	330
F	349
F #	370
G	392
G #	415
A	440
A #	466
B	494
C (above middle C)	523

Try this:

```
10 CALL SOUND(1000,440,0)
```

This plays "A above middle C" for one second.

These notes may be a little out of tune.

This is the reason:

only integers are used in the SOUND command
but decimal numbers are needed for notes in tune

and the frequency played may be off by up to 10% from the frequency asked for.

You get very high notes for pitch numbers above 5000.

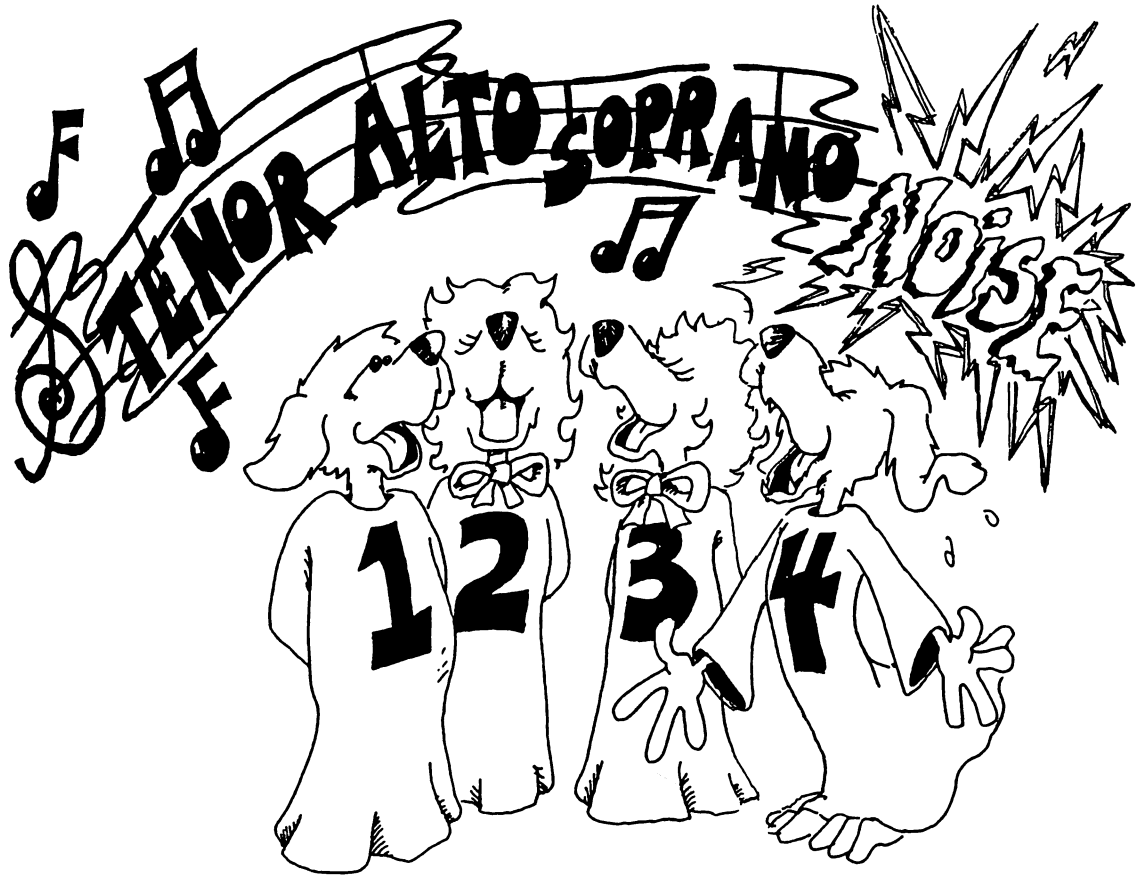
You may not be able to hear anything for pitch numbers above 10000.

LOUDNESS

Little numbers give loud sounds.

L = 30 sound is turned off
L = 10 sound is normal loudness
L = 0 sound is loudest

(Of course, you can turn up the TV to get a louder sound.)



ONE VOICE

```
10 REM ONE VOICE
20 PRINT "ONE VOICE"
22 PRINT "HOW LONG? <1 TO 4250>"
23 INPUT D
25 PRINT "WHAT PITCH <110 TO 10000>"
26 INPUT P
35 PRINT "HOW LOUD <0 TO 30>"
36 INPUT L
40 CALL SOUND( D, P, L)
45 FOR T=1 TO 1000
46 NEXT T
50 GOTO 20
```

DUET

```
10 REM DUET
20 PRINT "TWO VOICES"
25 PRINT "PITCH 1"
26 INPUT P1
30 PRINT "PITCH 2"
31 INPUT P2
40 CALL SOUND(2000,P1,0,P2,0)
45 PRINT
46 PRINT
50 GOTO 20
```

DOING TWO THINGS AT ONCE

The TI computer makes sounds and computes at the same time!

You can move characters on the screen while the sound is still going. Most other personal computers can't easily do this.

Try this:

```
10 CALL CLEAR
20 CALL SOUND(4250,110,0)
30 FOR I = 1 TO 15
33 PRINT I
36 NEXT I
40 CALL SOUND(4250,500,0)
45 PRINT "SECOND"
50 CALL SOUND(4250,1000,0)
55 PRINT "THIRD"
```

Line 20 asks for a low note. While it is still sounding, the loop in lines 30 to 33 is also running.

Then the program gets to line 40 where a new sound is called for. But the computer waits until the sound from line 20 is over before starting the new tone.

While line 40's sound is going, the computer prints in line 45 and then waits at line 50 for the second sound to end.

Finally, the sound of line 50 is made, and the program prints "THIRD" and then ends.

The computer prints:

```
** DONE **
```

while the sound is still going!

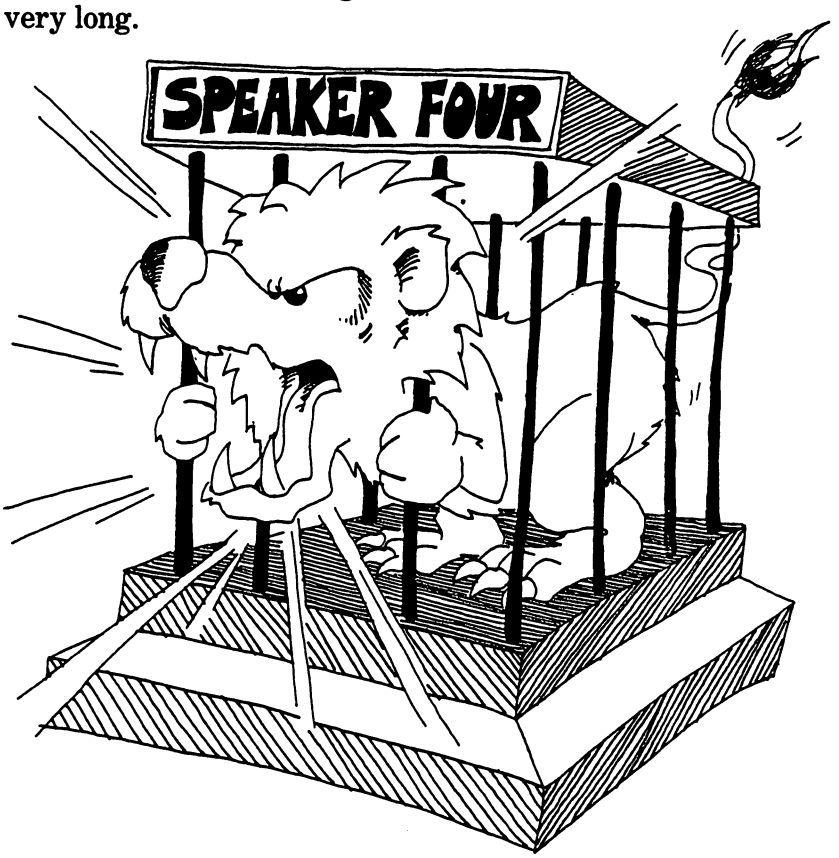
SOMETIMES IT IS POLITE TO INTERRUPT

If you want the new sound to start right away and not wait for the old sound to end, then put a minus sign in front of its duration number.

Change:

```
50 CALL SOUND(-4520,1000,0)
```

Run the program again and see that the high tone at the end cuts off the middle tone before it has run very long.



MAKING NOISES

You make noises by choosing a "pitch" that is one of these negative numbers:

−1	periodic noise type 1
−2	periodic noise type 2
−3	periodic noise type 3
−4	periodic noise mixed with tone 3
−5	"white" noise type 1
−6	"white" noise type 2
−7	"white" noise type 3
−8	"white" noise mixed with tone 3

Try this:

```
10 REM NOISE
20 FOR I=-1 TO -8 STEP -1
30 CALL SOUND(4000,I,0)
40 NEXT I
```

You are supposed to use noise −4 and −8 with one, two, or three other sounds.

Change:

```
30 CALL SOUND(4000,I,0,500,10,600,10,700,10)
```

COMPLICATED SOUNDS

You can try combinations to see what sounds you like:

```
10 REM COMBINATIONS
15 S1=200
16 S2=1000
20 FOR I=1 TO 20
22 S1=S1 * 1.1
24 S2=S2 * 0.95
30 CALL SOUND(100,S1,0,S2,0)
40 NEXT I
```

Assignment 20:

1. Make a list of sound effects. For each pitch number −1, −2, −3, −5, −6, and −7 tell what kind of sound you get. Try it togther with other voices and tones. Think of a game or program which the sound would be good for.
2. Make the sound of: a truck horn
   a laser gun
   an explosion
   a wind storm
3. Write a program to play a short tune, like "Row, Row, Row Your Boat" or "Mary Had A Little Lamb." Use a DATA statement to store the pitch numbers.

INSTRUCTOR NOTES 21 HI-RES GRAPHICS

TI BASIC has a powerful command, CALL CHAR( ), to define new characters.

It can be used to produce a new type font or make game characters. For example, 8 arrows, cars, planes, tanks, etc., pointing in different directions allow plenty of action in games. You can make ships, dinosaurs, and other elaborate high resolution graphics by using 2 or more adjacent characters.

A special character is made in two steps. First it is drawn on paper in an 8x8 grid. There is a page in this lesson that you should reproduce by office copier so your student will have plenty of grids to use.

Then, noticing that the grid is divided into 16 short rows of 4 dots each, you write a code word (16 characters long) that specifies the short rows in the character. You pick the code characters by referring to a table of 4-dot rows in the lesson.

Another description for those of you who are familiar with binary and hexadecimal notation is this: Each of the eight rows of the character is viewed as an eight digit binary number. Then the character is specified by writing one 16 digit hex number which is made up of the 8 (two digit) hex numbers specifying the rows of dots.

The color numbered "1" is called "transparent." It means that the dots are chosen to be of the "background" color which was specified by the CALL SCREEN(color) statement. (Defaults to green.) Transparent can be used either for the "off" or "background" dots of a character, or the "on" or "foreground" dots of the character.

Or even both. Then the character is a solid square of background color. This might be a way to use the CALL COLOR( ) statement to make invisible things on the screen appear and then disappear again!

QUESTIONS:

1. How many character sets are there?
2. How many characters are in each set?
3. What does "59" mean in the statement

```
40 CALL CHAR(59,"FFFF0000FFFF0000")
```

4. What does "FFFF0000FFFF0000" mean?
5. If you want the characters to sit on the "background" color instead of sitting in a little square of another color, what do you do?

LESSON 21 HI-RES GRAPHICS

Learn to make your own graphics characters for drawing "high-resolution" pictures.

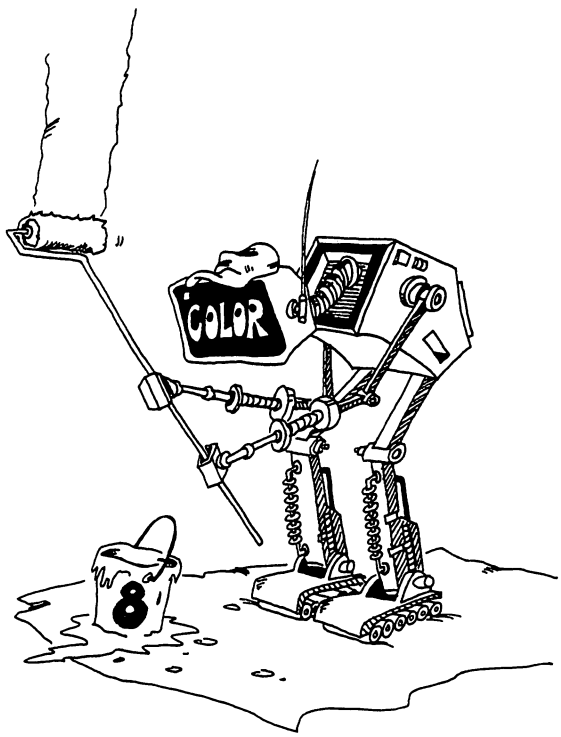
REVIEW OF GRAPHICS

Let us review the graphics you have already learned.

THE COLORS

1 TRANSPARENT	9 MEDIUM RED
2 BLACK	10 LIGHT RED
3 MEDIUM GREEN	11 DARK YELLOW
4 LIGHT GREEN	12 LIGHT YELLOW
5 DARK BLUE	13 DARK GREEN
6 LIGHT BLUE	14 MAGENTA (PURPLE)
7 DARK RED	15 GREY
8 CYAN (BLUE-GREEN)	16 WHITE

Adjust your TV controls to make the colors look right.



THE CHARACTER SETS

There are 16 sets. Each has 8 characters in it.

Set	character number							
1	32	33	34	35	36	37	38	39
2	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55
4	56	57	58	59	60	61	62	63
5	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79
7	80	81	82	83	84	85	86	87
8	88	89	90	91	92	93	94	95
9	96	97	98	99	100	101	102	103
10	104	105	106	107	108	109	110	111
11	112	113	114	115	116	117	118	119
12	120	121	122	123	124	125	126	127
13	128	129	130	131	132	133	134	135
14	136	137	138	139	140	141	142	143
15	144	145	146	147	148	149	150	151
16	152	153	154	155	156	157	158	159

THE COMMANDS

Color a character set using the CALL COLOR statement:

Enter:

```
10 REM GRAPHICS
15 CALL CLEAR
20 CALL COLOR(2,7,15)
```

This paints all 8 characters in set 2. They become red with a grey background.

Set 2 looks like this now,

Number	40	41	42	43	44	45	46	47
Character	(	)	*	+	,	-	.	/

but we will soon change some of the characters to look different. You put characters on the screen with CALL HCHAR and CALL VCHAR.

Add:

```
30 CALL HCHAR(3,6,42,4)
99 GOTO 99
```

Run the program. Line 30 means

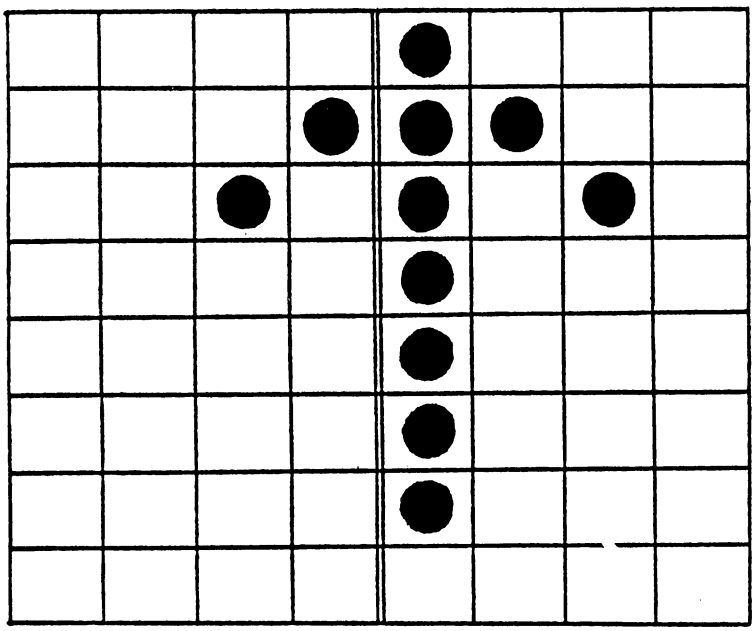
```
30 CALL HCHAR (row 3, col. 6, char. 42, repeat 4 times)
```

MAKING A NEW CHARACTER

Each character is a square made of dots.

The square is 8 dots across and 8 dots down.

Here is an arrow character:



0	8
1	C
2	A
0	8
0	8
0	8
0	8
0	0

You have to tell the computer two things:

which character number to use
which dots to light up

Let's change the "star" character to an arrow:

Add:

```
40 CALL CHAR(42,"081C2A0808080800")
41 CALL VCHAR(10,11,42,8)
```

Run the program.

When line 40 is executed, character 42, which used to be the star "*", becomes the arrow.

IMPORTANT! Even the stars already on the screen change to arrows!
Add:

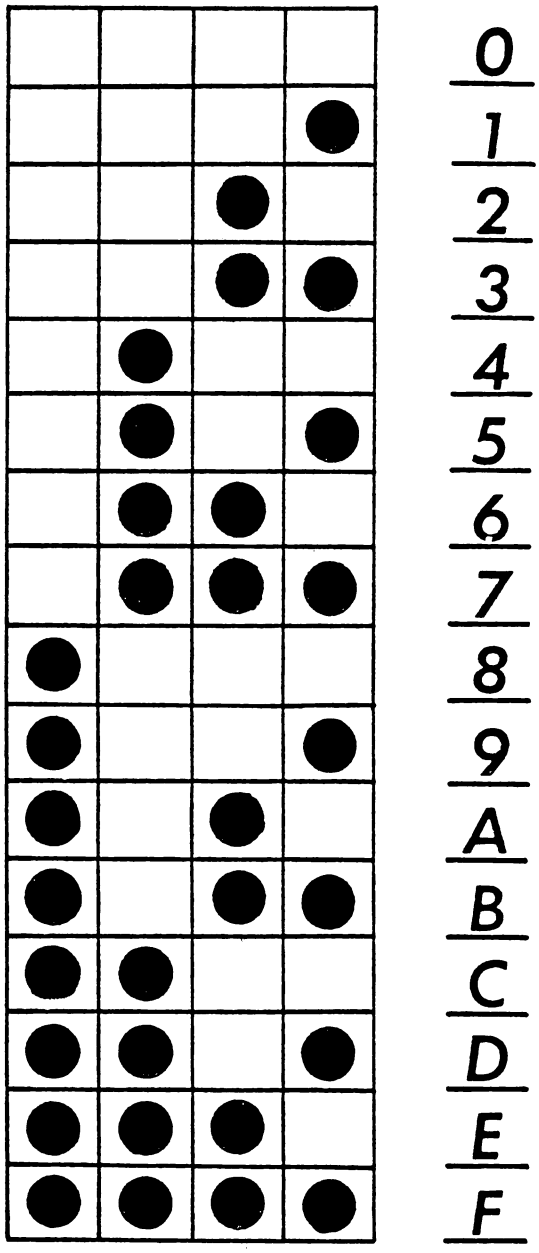
```
35 FOR T=1 TO 400
36 NEXT T
```

And run it again. Watch to see the "old" stars change to arrows.

CODE FOR THE CHARACTER

A special code tells the computer which dots are in the character.

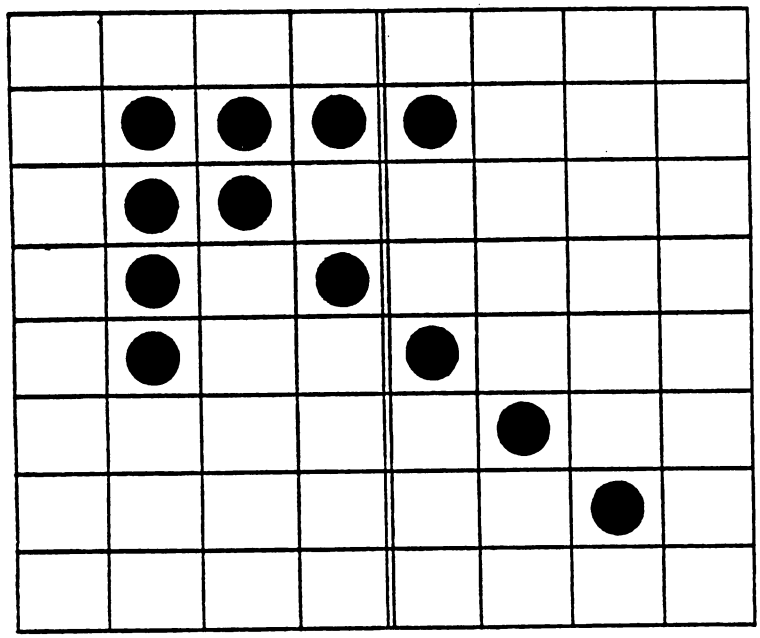
Each character is divided up into 16 little rows of four dots each. Each kind of 4-dot row has its own code name. The code is a number or a letter: 0 1 2 3 4 5 6 7 8 9 A B C D E F.



(If you know about binary and hexadecimal numbers, you can see why each little row is named the way it is.)

THE REST OF THE ARROWS

Here is a tilted arrow.



0	0
7	8
6	0
5	0
4	8
0	4
0	2
0	0

Assignment 21A:

1. Draw 6 more arrows, so you have 8. One arrow in each direction: up, down, left, right, and four tilted arrows in between the others.
2. For each arrow, write its code letters.
3. Add the above arrow characters to the above program, using CALL CHAR( ) command lines. Use all the numbers 40 to 48 which are in character set 2.
4. Add lines to the program so all the arrows on the screen revolve like little clock hands. Use CALL HCHAR and CALL VCHAR in a loop which has a delay loop nested inside it.
5. Now put the 8 arrows in 8 different character sets. Make each set a different color. When the arrows slowly revolve, they will also change color!

THE "TRANSPARENT" COLOR

Run:

```
10 REM FLASHING BACKGROUND
15 CALL CLEAR
20 CALL CHAR(42,"081C2A0808080800")
30 CALL COLOR(2,7,15)
31 CALL HCHAR(12,14,42)
35 FOR T=1 TO 400
36 NEXT T
40 CALL COLOR(2,7,1)
45 FOR T=1 TO 400
46 NEXT T
50 GOTO 30
```

The arrow sits in a little square that flashes white, then "transparent." (The green background shows through.)

Line 20 makes an arrow character and stores it in the memory.
Line 30 colors it red with a grey background.

30 CALL COLOR (char. 42, red color, white background)

Line 40 colors it red with a "transparent" background.

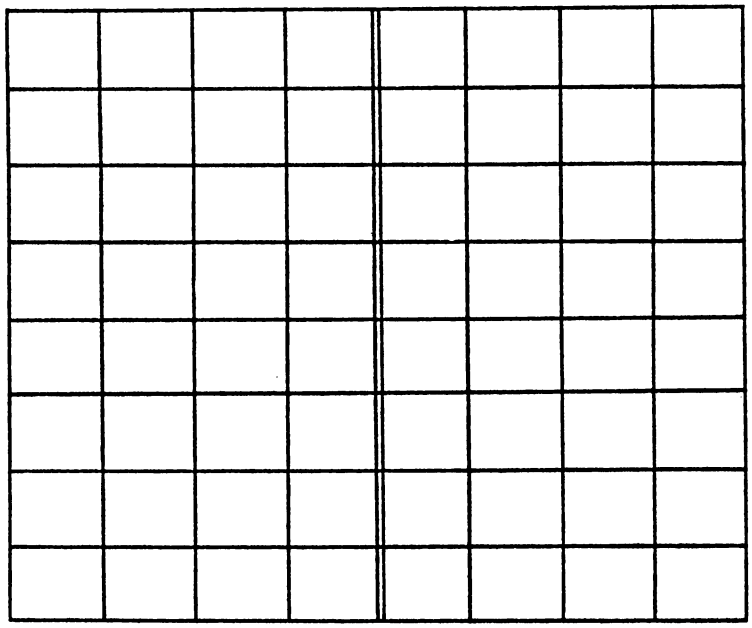
Change: `40 CALL COLOR(2,1,15)`

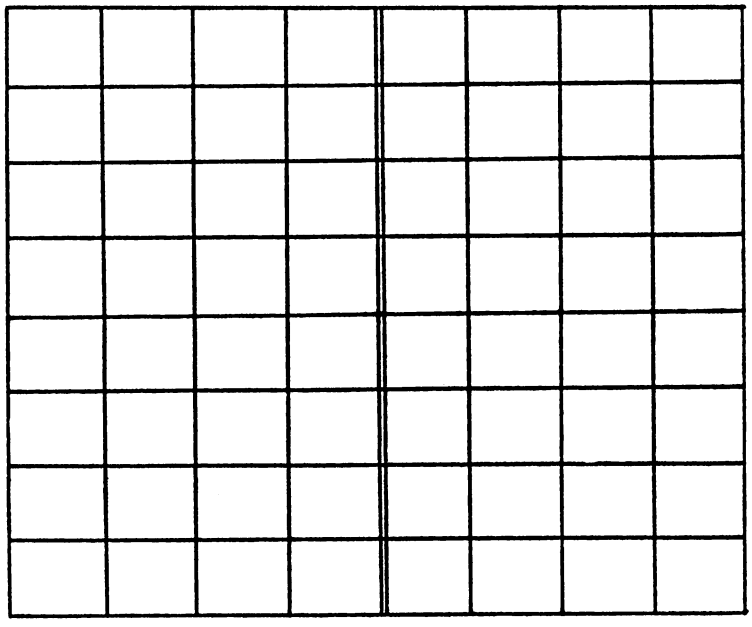
Run it again. What happens? .....

Assignment 21B:

1. Make some other characters which, seen from above, point in 8 different directions. They may be birds, cars, airplanes, tanks, etc. Draw them in copies of page 126 and figure out which code numbers they have.
2. Make the club, diamond, heart, and spade characters of a set of cards.

Do not draw on this page!
Instead make copies on an
office copier.





INSTRUCTOR NOTES 22 ASCII CODE, KEYBOARD, ON...GOTO

This lesson treats the ASCII code for characters and the functions ASC( ) and CHR$( ) which change characters to ASCII numbers and vice versa.

The ASCII code is primarily intended to standardize signals between hardware pieces such as computers with printers, terminals, other computers, etc. But within programs the ASCII numbers are also useful. The letters are numbered in increasing order and so the ASCII numbers are useful in alphabetizing routines. The numerical digits are also in order, and the punctuation marks also have ASCII numbers.

The CALL KEY statement gets keystrokes from the keyboard and reports them as ASCII numbers in a variable. This is treated in the next lesson.

The CALL CHAR statement treated later identifies characters by numbers from 30 to 159. The default value of these characters is that given by the ASCII code.

QUESTIONS:

1. Does ASC(S$) return a string or a number for its value?
2. Does ASC(S$) have a string or a number for its argument?
3. Same two questions for CHR$(N).
4. Which letter has the larger ASCII code number, B or W?
5. Do you know the ASCII code for the character "1"? Is it the number 1?
6. What will the computer do if you run this line:

```
10 PRINT CHR$(65); CHR$(30)?
```

   (If you don't know, try it.)

LESSON 22 ASCII CODE, KEYBOARD, ON...GOTO

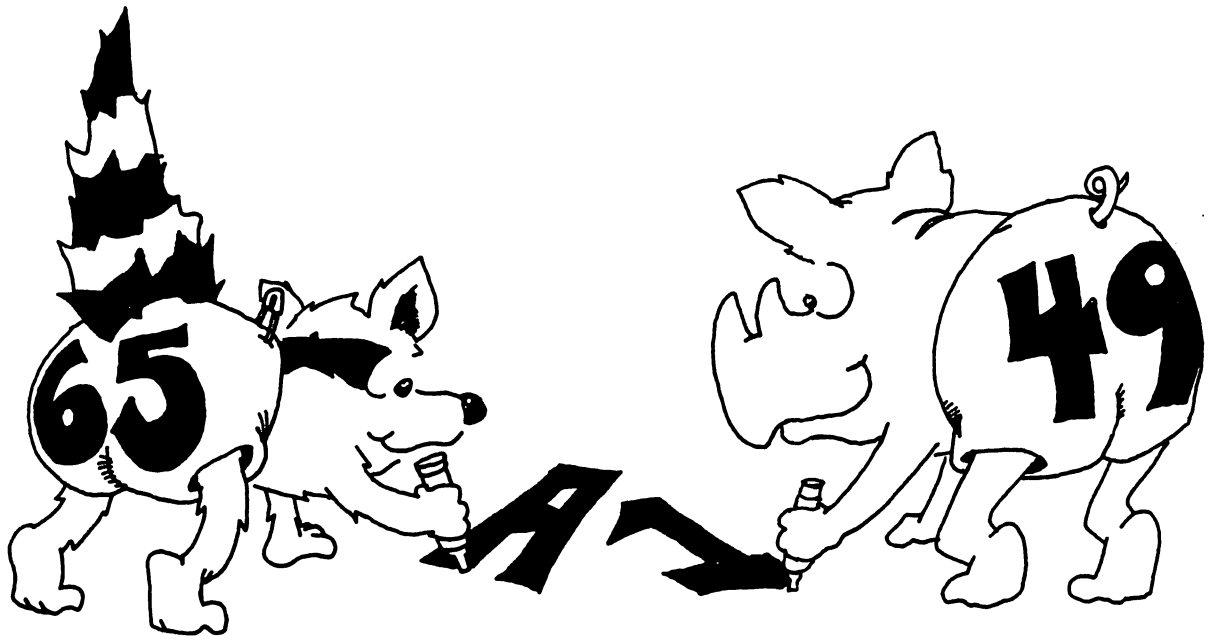
NUMBERING THE LETTERS IN THE ALPHABET

"That is easy," you say. "A is 1, B is 2, C is 3 ..."

Well, for some strange reason, it goes like this: A is 65, B is 66, C is 67 ... .

These numbers are called the ASCII code of the characters. ASCII is pronounced "ask-key."

The punctuation marks and number digits have ASCII code numbers too. Later you will learn how to make your own characters and give them ASCII numbers.



ASC( ) CHANGES CHARACTERS INTO NUMBERS

Use the ASC( ) function to change characters into ASCII numbers.

Run:

```
10 REM *** WHAT NUMBER IS THIS KEY? ***
20 PRINT "PRESS KEYS TO SEE ASCII NUMBER"
30 INPUT C$
40 PRINT C$;TAB(5);ASC(C$)
50 GO TO 30
```

Try out some letters, digits, and punctuation.

Press FCTN CLEAR to end the program. Then SAVE it to tape.

CHR$( ) CHANGES NUMBERS INTO CHARACTERS

Use CHR$( ) to change ASCII code numbers into a string holding one character.

Run:

```
10 REM /// DISPLAY ASCII ///
11 REM
20 CALL CLEAR
30 FOR I=30 TO 127
40 PRINT I, CHR$(I)
50 FOR T=1 TO 200
51 NEXT T
60 NEXT I
```

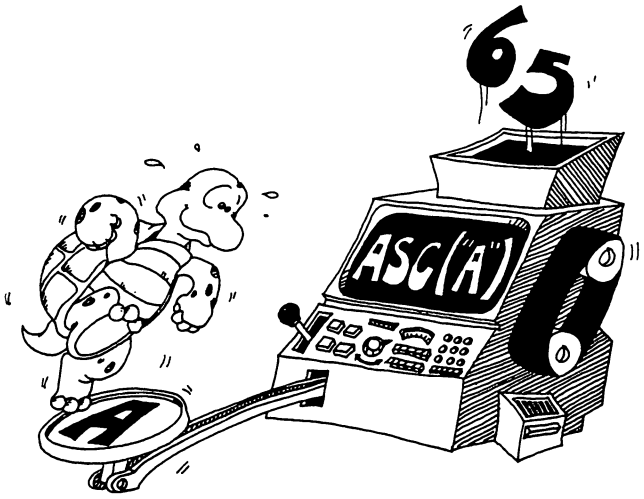
Save the program to tape.

CHR$( ) IS THE REVERSE OF ASC( )

We showed these two functions: ASC( ) and CHR$( ).

ASC( ) gives you the ASCII number for the FIRST character in the string.

CHR$( ) does the reverse. It gives you the character belonging to each ASCII number.



THE ASCII NUMBERS FOR CHARACTERS

Here are the groups of characters and their ASCII numbers:

13			ENTER key
30			cursor
31			edge character (invisible)
32	to	47	punctuation
47	to	57	number digits
58	to	64	punctuation
65	to	90	capital letters
91	to	96	more punctuation
97	to	126	small letters
127			DEL (invisible)
128	to	:::	free for graphics use

ALPHABETICAL LIST

What good are the ASCII numbers? They are needed for the CALL KEY command explained in the next lesson.

They can also help in making alphabetical lists.

Run:

```
10 REM ALPHABETIZE
20 PRINT
30 INPUT"GIVE ME A LETTER: ":A$
35 PRINT
40 INPUT"GIVE ME ANOTHER: ":B$
45 A=ASC(A$)
46 B=ASC(B$)
47 REM PUT IN ALPHABETICAL ORDER BY
48 REM SEEING WHICH HAS THE LOWER ASCII NUMBER.
50 IF A<B THEN 60
51 REM SWAP THE LETTERS
52 X=A
53 A=B
54 B=X
60 PRINT
65 PRINT"HERE THEY ARE IN ALPHABETICAL ORDER"
70 PRINT
71 PRINT CHR$(A);TAB(5);CHR$(B)
```

Save it to tape.

THE ON...GOTO COMMAND

The SNAKE program below uses the ON ... GOTO command.

```
125 ON Z GOTO 130,135,140,145

if   Z    is      1          GOTO  130
                  2                135
                  3                140
                  4                145
if   Z    is something else   print:
                  * BAD VALUE IN 125
```

After the GOTO you can put in one, two, or as many numbers as you want. Each number is the same as the number of a line somewhere in the program.

Assignment 22:

1. Write a program which asks for a word. Then it rearranges all the letters in alphabetical order.
2. Write a program which speaks "double dutch." It asks for a sentence, then removes all the vowels and prints it out.
3. Write a program which uses CALL KEY to get a letter A to C to use in a menu. Change the letter to a number 1 to 3. Then use the ON...GOTO command to pick which menu item to do.

```
2 REM +++++ SNAKE +++++
3 GOTO 1000
100 REM
101 REM ----------------MAIN LOOP
102 REM
109 REM ----------------GET KEY STROKE
110 CALL KEY(0,W,S)
111 IS S=0 THEN 125
113 REM ----------------TURN WHICH WAY?
114 IF W<>46 THEN 120
116 Z=Z-1
117 IF Z<>0 THEN 125
118 Z=4
119 GOTO 125
120 Z=Z+1
121 IF Z<5 THEN 125
122 Z=1
124 REM ---------------NEW POSITION OF HEAD
125 ON Z GOTO 130,135,140,145
130 Y=Y-1
131 GOTO 150
135 X=X-1
136 GOTO 150
140 Y=Y+1
141 GOTO 150
145 X=X+1
149 REM ---------------SNAKE MOVES
150 A=B
151 B=C
152 C=D
153 D=E
154 E=F
155 F=X
160 L=M
161 M=N
162 N=O
163 O=P
164 P=Q
165 Q=Y
169 REM ---------------ERASE OLD TAIL END
170 CALL HCHAR(L,A,32,1)
171 REM ---------------PRINT NEW HEAD
172 CALL HCHAR(Y,X,50,1)
199 GOTO 100
1000 REM
1001 REM +++++ S N A K E +++++
1002 REM
1500 GOSUB 3000
```

```
2000 REM
2001 REM BORDER
2002 REM
2010 CALL CHAR(42,"FFFFFFFFFFFFFFFF")
2015 CALL CHAR(59,"FFFFFFFFFFFFFFFF")
2020 CALL COLOR(2,5,1)
2021 CALL COLOR(3,9,1)
2030 CALL CLEAR
2099 REM --------------MAKE BORDER
2100 CALL HCHAR(1,1,42,32)
2101 CALL HCHAR(24,1,42,32)
2110 CALL VCHAR(1,1,42,24)
2111 CALL VCHAR(1,32,42,24)
2199 REM --------------SNAKE EGG IN CENTER
2200 X=16
2201 Y=12
2210 A=X
2211 B=X
2212 C=X
2213 D=X
2214 E=X
2215 F=X
2220 L=Y
2221 M=Y
2222 N=Y
2223 O=Y
2224 P=Y
2225 Q=Y
2300 Z=1
2999 GOTO 100
3000 REM
3001 REM ----------------------INSTRUCTIONS
3002 REM
3005 CALL CLEAR
3010 PRINT "TURN LEFT, '<' KEY"
3020
3030 PRINT "TURN RIGHT, '>' KEY"
3100 FOR T=1 TO 2000
3101 NEXT T
3999 RETURN
```

INSTRUCTOR NOTES 23 SECRET WRITING AND CALL KEY

CALL KEY is a method of requesting a single character from the keyboard. The computer polls the keyboard and reports two things: a keystroke and a status.

There is no screen display at all. No prompt or cursor is displayed while waiting, and the keystroke, when made, is not echoed to the screen.

The utility of the CALL KEY command lies just in this fact. For example, a secret password may be received with a series of CALL KEY's without displaying it to bystanders.

Another advantage over INPUT is that no ENTER key pressing is required. This makes CALL KEY useful in "user friendly" programming.

The CALL KEY command doesn't wait for a key to be pressed. This makes it useful in action games. If you need to have the program wait for a keystroke, you must do an IF and branch back until a keystroke is detected. This is demonstrated in the lesson.

Along with saving the keystroke in a variable, the command also saves a "status" which records if this is the same or a different key stroke from the last time. It also tells whether a key is presently down or not.

If you want to get numerical values, get them as strings and convert them to numbers using the VAL( ) function discussed in a later lesson.

QUESTIONS:

1. Compare INPUT and CALL KEY. One gets one letter at a time, the other gets whole words and sentences. One has a cursor, the other not. One prints on the screen, the other not. One needs the ENTER key, the other not. Which does which?

LESSON 23 SECRET WRITING AND CALL KEY

THE INPUT STATEMENT

Examples:

```
10 INPUT A$
10 INPUT N
10 INPUT NAME$,AGE,DAY,MONTH$,YEAR
```

The computer waits for you to type a word, sentence or number.

Then you press the ENTER key to tell the computer that you are done entering.

THE CALL KEY STATEMENT

The CALL KEY statement is different from INPUT.

It doesn't wait.

It looks to see if a key is being pressed. If so, it puts the ASCII number of the key into a numerical variable box.

You do not have to press ENTER.

CALL KEY FOR INVISIBLE TYPING

Nothing shows on the screen:

no question mark will show
no cursor will show
what you type will not show.

To see what happens, you have to PRINT the variable.

Run:

```
10 CALL KEY(0,K,S)
20 PRINT K
25 REM BOX K HOLDS THE ASCII NUMBER
30 GOTO 10
```

The computer prints −1 until you press a key. Then it prints the ASCII number of the character.

Try this: Hold down the "A" key

See that the computer prints "65" which is the ASCII number of the letter "A".

Try holding down different keys.

If you press a key too quickly, the computer may miss it. Try it!

MAKING THE COMPUTER WAIT FOR YOU TO TYPE

Add to the above program:

```
15 IF K=-1 THEN 10
20 PRINT K;TAB(7); CHR$(K)
```

Line 15 makes the computer keep looking until a key is pressed.

SAME KEY OR NEW KEY?

The CALL KEY(0,K,S) statement fills two variable boxes, K and S.

In the K box it puts the ASCII number of the key being pressed right now. (It puts −1 if no key is being pressed.)

What it puts into the S box depends on what key was pressed the time before.

In the S box it puts:

S = 0 if no key is being pressed now
S = 1 if a new key is being pressed
S = −1 if the same key as before is being pressed

Try this:

```
10 CALL KEY(0,K,S)
20 PRINT CHR$(K);S
25 FOR T=1 TO 200
26 NEXT T
30 GOTO 10
```

Press some keys and see when S is zero, −1 or 1.

SECRET WRITING

Use CALL KEY in guessing games for entering the word or number to be guessed without the other player being able to see it.

Run this program:

```
10 REM -----CALL KEY-----
20 CALL CLEAR
30 PRINT "PRESS ANY KEY"
40 CALL KEY(0,K,S)
41 IF K = -1 THEN 40
45 CALL SOUND(300,900,10)
47 FOR T=1 to 1000
48 NEXT T
50 PRINT "THE KEY YOU PRESSED WAS "CHR$(K)
99 GOTO 40
```

Run this one too:

```
10 REM *** BACKWARDS ***
20 CALL CLEAR
30 PRINT"TYPE IN A 5 LETTER WORD"
35 PRINT
40 FOR I= 1 TO 5
42 CALL KEY(0,L,S)
43 IF L = -1 THEN 42
44 W$=CHR$(L) & W$
46 CALL KEY(0,L,S)
47 IF S <> 0 THEN 46
48 NEXT I
50 PRINT"NOW HERE IT IS BACKWARDS"
55 PRINT
60 PRINT W$
```

Line 43 will not let the program continue until a key is pressed.

Lines 46 and 47 make sure that the key was let up before line 42 can ask about a new key. Try the program without lines 46 and 47 and see how your letters repeat!

MAKING WORDS OUT OF LETTERS

The CALL KEY command gets one letter at a time. To make words, glue the strings.

```
10 REM GET A WORD
20 CALL CLEAR
30 PRINT"TYPE A WORD. END IT WITH AN 'ENTER'."
35 W$=""
40 CALL KEY(0,L,S)
41 IF L = -1 THEN 40
50 IF L = 13 THEN 80
60 W$=W$ & CHR$(L)
62 CALL KEY(0,L,S)
63 IF S<>0 THEN 62
65 GOTO 40
80 REM WORD IS FINISHED
85 PRINT W$
```

How does the computer know when the word is all typed in? Line 40 looks to see if the ENTER key was pressed. The ASCII number of the ENTER key is "13". Line 50 branches to print the word if the ENTER key was pressed.

CALL KEY FOR NUMBERS

If you want to enter a secret number from the keyboard, you have to enter digit characters (0 to 9), glue them into a string, and then use the VAL function explained in Lesson 27.

Assignment 23:

1. Write a program which has a "menu" for the user to choose from. The user makes her choice by typing a single letter. Use CALL KEY to get the letter. Example:

```
PRINT "WHICH COLOR? <R=RED, B=BLUE, G=GREEN>"
```

2. Write a sentence making game. Each sentence has a noun subject, a verb, and an object. The first player types a noun (like "The donkey"). The second player types a verb (like "sings"). The third player types another noun (like "the toothpick."). Use CALL KEY so no player can see the words of the others. You may expand the game by having adjectives before the nouns.

INSTRUCTOR NOTES 24 PRETTY PROGRAMS, GOSUB, RETURN

This lesson covers subroutines.

Like GOTO, GOSUB causes a jump to another line number. The only difference is that in GOSUB the computer stores the next line number following the GOSUB on a stack. When the computer encounters a RETURN statement, it pops the line number off the stack and returns control to that line.

Subroutine calls can be nested at least 9 deep.

The END command can be put anywhere in the program and you can use as many end statements as you wish. All that END does is to return control to the command mode.

Subroutines are useful not only in long programs but in short ones where "chunking" the task into sections leads to clarity.

GOSUB was put into BASIC for making modules. This lesson shows modular construction in a graphics program. The same subroutine which writes the letter "J" also erases it.

The JUMPING J exercise allows the student to try many different effects in the moving graphics display.

QUESTIONS:

1. What happens when the command END is executed?
2. How is GOSUB different from GOTO?
3. What happens when RETURN is executed?
4. If RETURN is executed before GOSUB, what happens?
5. What does "call the subroutine" mean?
6. How many END commands are you allowed to put into one program?
7. Why do you want to have subroutines in your program?

LESSON 24 PRETTY PROGRAMS, GOSUB, RETURN

Run this program then save it to tape:

```
100 REM MAIN PROGRAM
101 REM
110 PRINT "HOP TO THE SUBROUTINE"
120 GOSUB 200
130 PRINT "BACK FROM THE SUBROUTINE"
133 PRINT
135 PRINT "HOP AGAIN"
140 GOSUB 200
150 PRINT "BACK AGAIN"
190 END
199 REM
200 REM SUBROUTINE
201 REM
210 PRINT "IN THE SUBROUTINE"
212 CALL SOUND(500,300,100)
215 FOR T=1 TO 1000
216 NEXT T
220 PRINT "PACK YOUR BAGS, BACK WE GO"
290 RETURN
```

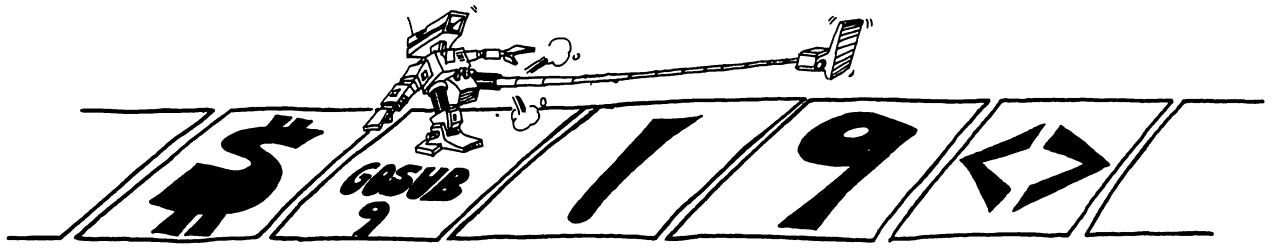
This is the skeleton of a long program. The main program starts at line 100 and ends at line 190.

Where there are PRINT commands, you may put in many more program lines.

Line 120 and line 140 "call the subroutine." This means the computer goes to the lines in the subroutine, does them, and then comes back.

The GOSUB 200 command is like a GOTO 200 command except that the computer remembers where it came from so that it can go back there again.

The RETURN command tells the computer to go back to the next statement after the GOSUB.



WHAT GOOD IS A SUBROUTINE?

In a short program, not much.

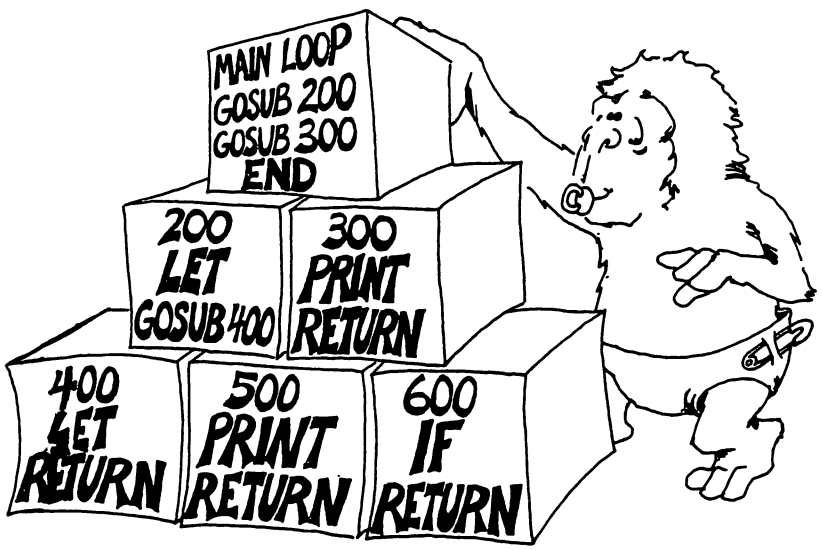
In a long program, it does two things:

1. It saves you work. It saves space in memory. You do not have to type in the same program lines in different parts of the program.
2. It makes the program easier to understand and faster to write and debug.

Assignment 24A:

1. Write a short program which uses subroutines. It doesn't have to do anything useful, just print some silly things. In it put three subroutines:

   Call one of them twice from the main program.
   Call one of them from another of the subroutines.



MOVING PICTURES

```
10 REM ↑↑↑ JUMPING J ↑↑↑
20 CALL CLEAR
22 X=15
23 Y=12
24 D=1
25 FOR J=1 TO 5
26 FOR I=1 TO 5
30 CH=42
31 GOSUB 100
40 CH=32
41 GOSUB 100
45 Y=Y-D
50 NEXT I
55 D=-D
60 NEXT J
90 END
100 REM
101 REM === DRAW THE J ===
102 REM
110 CALL HCHAR(Y,X,CH,5)
120 CALL VCHAR(Y+1,X+2,CH,7)
130 CALL HCHAR(Y+7,X,CH,2)
140 CALL HCHAR(Y+6,X,CH)
190 RETURN
```

The picture is the letter "J." The subroutine starting in line 100 draws the "J." Before you GOSUB 100 you pick what character you want the "J" to be. Look at line 30 and at line 40. If you pick the space character, then the subroutine erases a "J" from that spot.

The subroutine draws the "J" with its upper left corner at the spot X,Y on the screen. When you change X or Y (or both) the "J" will be drawn in a different spot. Lines 22 and 23 say that the first "J" will be drawn near the middle of the screen.

The variable "D" tells how far the "J" will move from one drawing to the next. Line 24 makes "D" equal to 1, but line 55 changes D to -1 after 5 pictures have been drawn. A negative "D" makes the picture move down.

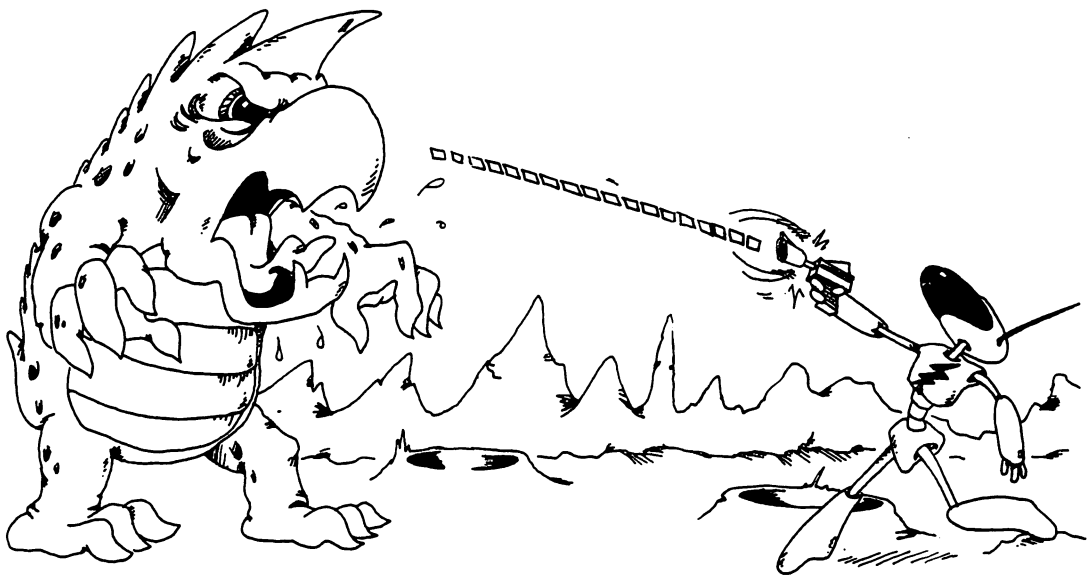
Line 45 says that each picture will be drawn at the spot where Y is smaller than the last Y by the amount D.

Assignment 24B:

1. Enter the JUMPING J program and run it. Then make these changes:
2. Change the subroutine so it prints your own initial.
3. Change the character to a solid square. Make its color blue.
4. Change the "jumping" to "sliding" (so the initial moves horizontally instead of vertically).
5. Change the starting point to the lower right hand corner instead of the middle of the screen.
6. Change the distance the slide goes to 10 steps instead of 5.
7. Change the size of each step from 1 to 2.
8. Change the "sliding" so it slides uphill. Use

```
X=X+D
Y=Y-D
```

9. Change the program so the initial changes color from green (color 3) through all the colors to white (color 16) as it jumps.



INSTRUCTOR NOTES 25 LINE EDITING

Line editing allows you to call up a line from a program and change it.

```
EDIT 125
```

You can make any changes you wish, except the line number must remain the same. This restriction makes the EDIT mode less useful than it might be.

Editing a line is conceptually more complicated than simply typing it over again. However, most students will learn the editing procedure and use it for most line repairing.

The rules for naming variables in TI BASIC are rather free from restrictions compared to some other versions of BASIC. The name can have up to 15 characters. Names must start with a letter character. You cannot use a reserved (or "key") word as a name. There are certain punctuation marks you may not put in a name. In fact, it is better to leave all punctuation out of names (except the "$" at the end of string names).

QUESTIONS:

1. How do you put an "old line" on the screen to fix it?
2. If you want to copy a line but give it another number, can you use the EDIT command?
3. After you have fixed the line, how do you get it back into memory?

LESSON 25 LINE EDITING

PRACTICE MOVING THE CURSOR

Remember how to move the cursor left and right?

FCTN and left arrow key (on "S")

FCTN and right arrow key (on "D")

Push the keys once to move one space.

Hold down the keys and the cursor will keep moving.

FIXING A LINE

You learned this before.

To fix a line that you are typing:

1. Use the arrow keys to move the cursor to the error.
2. Fix things up:

   type the correct letter
   or use the FCTN DEL keys to remove (delete) a letter
   or use the FCTN INS keys to stick in (insert) letters.
3. Press ENTER to put the line in memory.

LINE EDITING: CHANGING A LINE THAT IS IN MEMORY

Sometimes you don't find the error until after the line is put in memory.

Sometimes you just change your mind about the line.

Do you have to type the line over? No!

Suppose you want to change line 623. Do these three things:

1. Enter EDIT 623 to put the line on the screen.
2. Move the cursor to the error in the line using the FCTN and the left arrow keys.
3. Fix the line. Press the ENTER key when done.

PRACTICE LINE EDITING

Enter:

```
623 PRINT "I LIKE SPINACH"
CALL CLEAR
```

Now enter:

```
EDIT 623
```

Press FCTN and the right arrow to move the cursor over the "L" in "LIKE."

Press the FCTN INS keys. Type `DON'T`. This fixes the line to be

```
"I DON'T LIKE SPINACH"
```

Press ENTER.

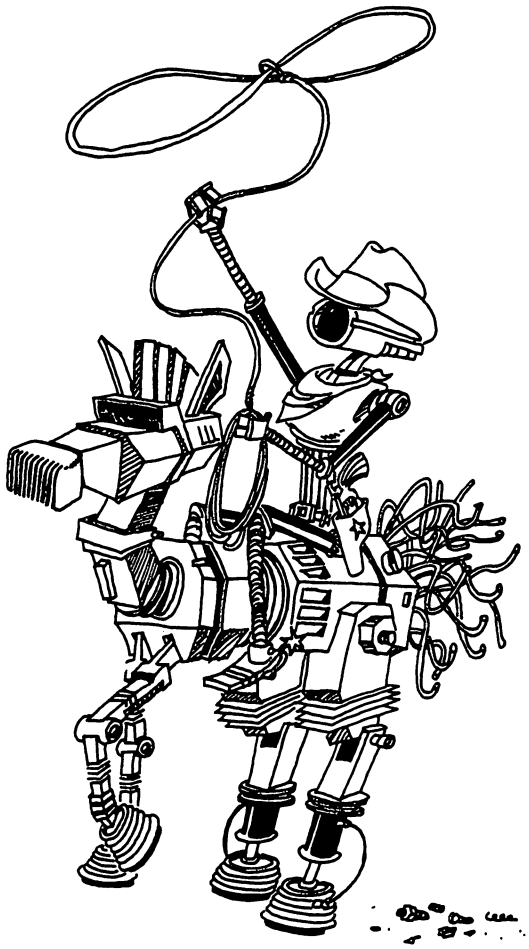
Then LIST 623 to see if the line is correct.

VARIABLE NAMES

Variable names can be up to 15 characters long. They must start with a letter and have only letters and numbers in them. (Except, of course, string variables must end in a dollar sign.)

(Actually, certain punctuation characters are allowed inside names, but you will make fewer errors if you do not allow any.)

Good names:	`JOE`	
	`T9`	
	`WX$`	
	`REDCOW$`	
	`FATLETTER1`	
	`COWBOY`	
Bad names:	`2SIDE`	starts with a number
	`X%`	has punctuation char.
	`LET`	is a keyword



The computer will print:

```
* BAD NAME
```

if you try to enter a line that has an incorrect name in it.

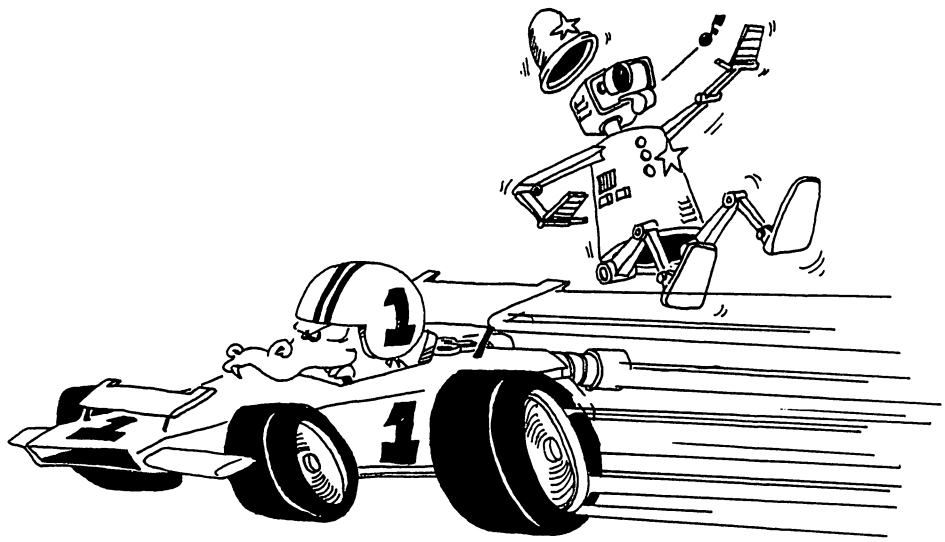
It is a good idea to make your variable names describe the variable. Examples:

CAR1 and CAR2	in a racing game
HOUSE and HOTEL	in a board game
GUESS$ and COLOR$	in a color guessing game

On the other hand, short names are quicker to type.

Assignment 25:

1. Load one of your old programs from tape and practice EDITing lines.



INSTRUCTOR NOTES 26 SNIPPING STRINGS: SEG$, LEN, POS

In this lesson the functions:

SEG$ LEN POS

are demonstrated.

These functions together with the concatenation operation "&" allow complete freedom to cut up strings and glue them back in any order.

The SEG$( ) function is similar to the MID$( ) of some other dialects of BASIC. As such, it can do the job of RIGHT$( ) and LEFT$( ) too.

The LEN function just returns the number of characters in its string argument.

TI BASIC uses the "&" for concatenation of strings. Other BASIC dialects often use "+." The latter is more confusing, because it is also used in arithmetic.

The POS statement is a nice feature of TI BASIC. It allows you to search for one string inside of another string. It reports the position of the first letter of the first occurrence of the string. The search starts at a given "starting number."

QUESTIONS:

1. If you want to save the "STAR" from "STARS AND STRIPES," what function will you use? What arguments?
2. If you want to save "AND," what function and arguments?
3. If you want to count the number of characters in the string PQ$, what function do you use? What argument?
4. What is wrong with each of these lines?

```
10 A$=LEFT$(4,D$)
10 RIGHT$(R$,I)
10 F$=SEG$(A,3)
10 J$=LEFT(R$,YT)
```

5. What command will search a string PQ$ to find where "THE" is in it?
6. Write a short program which takes the word "computer" and makes it into "putercom."

LESSON 26 SNIPPING STRINGS: SEG$, LEN, POS

GLUING STRINGS

You already know how to glue strings together:

Example:

```
55 A$="CON" & "CAT" & "EN" & "ATION"
60 PRINT A$
```

The real name for "gluing" is "concatenation."

Concatenation means "make a chain." Maybe we should call them "chains" instead of "strings."

SNIPPING STRINGS

Let's cut a piece off a string. Enter and run:

```
10 REM >>> SCISSORS >>>
20 CALL CLEAR
30 N$="123456789"
35 Q$=SEG$(N$,3,4)
40 PRINT Q$, N$
```

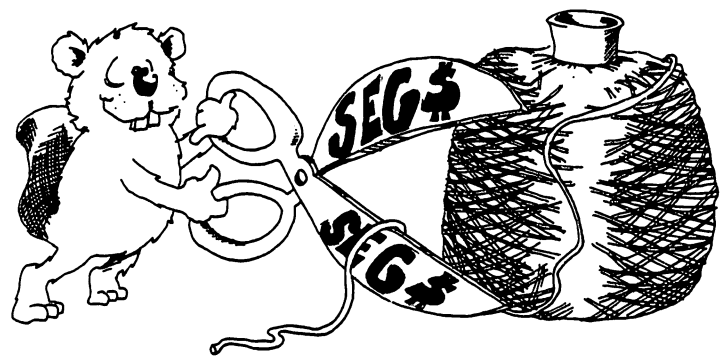
The SEG$ function snips out a piece of the string. The snipped off piece can be put into a box or printed or whatever.

Here is what line 35 does:

Get the string from box N$.
Count over 3 letters and start saving letters into box Q$.
Save 4 letters.

Rule: The SEG$( ) function needs three things inside the ( ) signs.

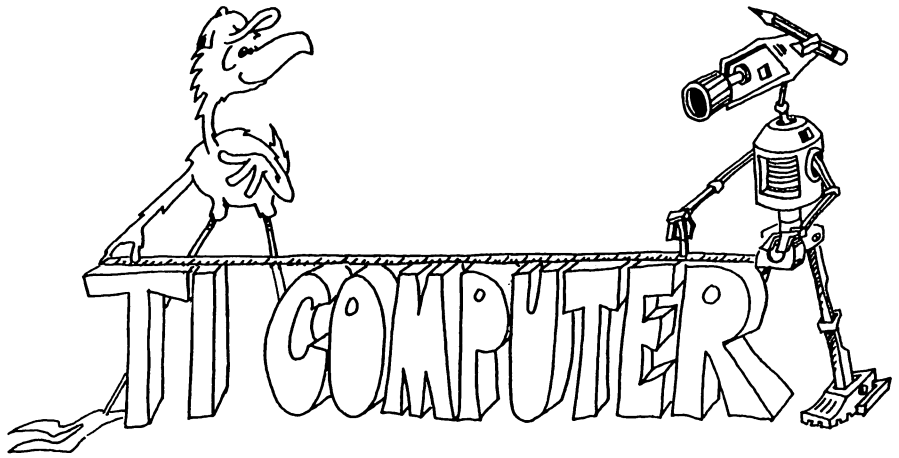
The string you want to snip.
The number of the first character
The number of characters to snip out



MORE SNIPPING AND GLUING

The pieces of string you snip off can be glued back together in a different order.

Run:
```
10 REM ::: SCISSORS AND GLUE :::
20 CALL CLEAR
30 N$="123456789ABCDEF"
35 FOR I=1 TO 13
40 L$=SEG$(N$,I,3)
42 M$=SEG$(N$,14-I,3)
45 Q$=M$ & L$
50 PRINT Q$
60 NEXT I
```



HOW LONG IS THE STRING?

Run:
```
10 REM ::: LONG ROPE :::
20 CALL CLEAR
30 PRINT "GIVE ME A STRING: "
31 INPUT N$
40 L=LEN(N$)
50 PRINT "THE STRING: '";N$;"'"
55 PRINT
56 PRINT "IS ";L;" CHARACTERS LONG"
```

The function LEN( ) tells the number of characters in the string. It counts everything in the string, even the spaces.

LOOK MA, NO SPACES

Enter:
```
10 REM <<< NO SPACES >>>
20 PRINT
21 PRINT
30 PRINT"GIVE ME A LONG SENTENCE"
31 PRINT
35 INPUT S$
40 L=LEN(S$)
45 T$=""
49 REM --------------LOOK AT EACH CHAR.
50 FOR I=1 TO L
60 L$=SEG$(S$,I,1)
70 IF L$=" " THEN 90
71 REM --------------SKIP SPACES
72 T$=T$ & L$
90 NEXT I
92 PRINT
94 PRINT "HERE IT IS WITH NO SPACES:"
96 PRINT
98 PRINT T$
```

Line 60 snips just one letter at a time out of the middle of the string.

LOOKING FOR A WORD IN A SENTENCE

The POS( ) function tells where one (short) string is located in another (longer) string.

"POS" is short for "position."

Run:
```
10 REM WORM
15 A$="CAT RAT DOG HORSE MOUSE BIRD WORM AARDVARK
TURTLE FISH CALERPITTER"
20 Z=POS(A$,"WORM",1)
30 B$=SEG$(A$,Z,4)
40 PRINT B$, Z
```

Line 15 A long string is put into box A$.
Line 20 The POS( ) function looks for "WORM" in A$
Line 30 The "WORM" is snipped out of the A$
Line 40 And PRINTed.

POS( ) is a function. It "returns a value." It works like this:

POS(long string, short string, start at number)

The short string is supposed to be somewhere inside the long string.

You start looking at the "start number." You usually will start at the beginning of the "long string," so the "start number" will usually be 1.

Then the computer counts letters starting from the left until it gets to the first letter of the "short string." It "returns" to the expression with the number at which the short string started.

Assignment 26:

1. Change the WORM program so that it asks the user which animal to look for. Then use LEN to find the length of the name. Finally, the animal name must be found in the string A$.
2. Write a secret cipher making program. You give it a sentence and it finds how long it is. Then it switches the first letter with the second, third with the fourth, etc. Example:

   THIS IS A DRAGON becomes:

   HTSII S ARDGANO
3. Write a question answering program. You give it a question starting with a verb and it reverses verb and noun to answer the question. Example:

   ARE YOU A TURKEY?

   YOU ARE A TURKEY.
4. Write a PIG LATIN program. It asks for a word. Then it takes all the letters up to the first vowel and puts them on the back of the word, followed by AY. If the word starts with a vowel, it only adds LAY. Examples:

   BOX becomes OXBAY

   APPLE becomes APPLELAY

INSTRUCTOR NOTES 27 SWITCHING NUMBERS WITH STRINGS

This lesson treats two functions, STR$ and VAL. A general review of the concept of function is also made.

STR$ takes a number and makes a string that represents it.

VAL does just the opposite, taking a string and making a numerical value from it. If the string does not represent a number (for example "5T7") then the computer prints:

```
* BAD ARGUMENT IN 10
```

The interconversion of the two main types of variables adds great flexibility to programs involving numbers.

You can slice up a number and rearrange its digits by first converting it into a string. This is demonstrated in the assignment which makes a number "march" by repeatedly putting its rear digit in the front.

Functions "return a value" to the expression they are in. One also says that functions are "called" just as one "calls" a subroutine. The reason is, of course, that functions are implemented as subroutines on the machine code level.

QUESTIONS:

1. If your number "marches" too quickly in the program of assignment 27, how do you slow it down?
2. If your program has the string "GEORGE WASHINGTON WAS BORN IN 1732." write a few lines to answer the question "How long ago was Washington born?" (You need to get the birthdate out of the string and convert it to a number.)
3. What is a "value"? What is meant by "a function returns a value"? What are some of the things you can do with the value?
4. What is an "argument" of a function? How many arguments does the SEG$( ) function have? How many for the CHR$( ) function?
5. Can you put a function at the start of a line?

LESSON 27 SWITCHING NUMBERS WITH STRINGS

This lesson explains two functions: VAL( ) and STR$( ).

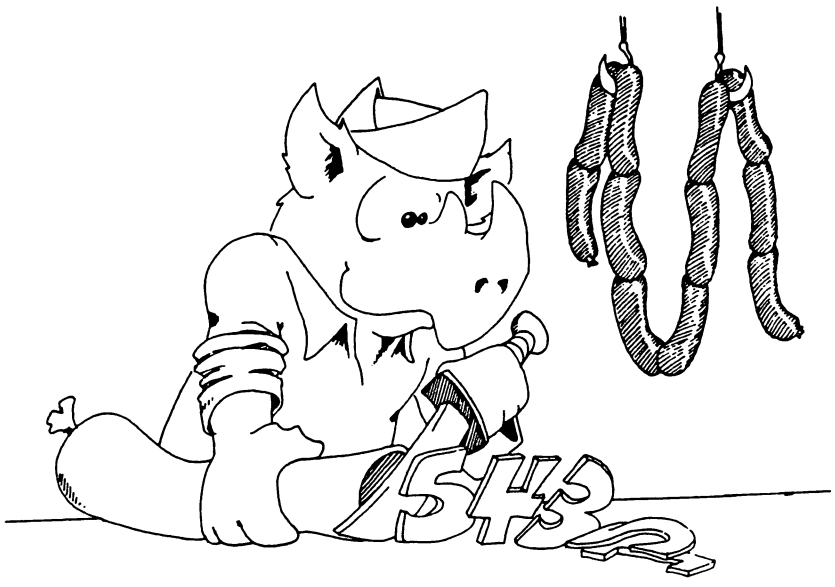
MAKING STRINGS INTO NUMBERS

We have two kinds of variables, strings and numbers. We can change one kind into the other.

Run:

```
10 REM MAKING STRINGS INTO NUMBERS
20 CALL CLEAR
30 L$="123"
40 M$="789"
50 L=VAL(L$)
60 M=VAL(M$)
70 PRINT L
72 PRINT M
74 PRINT "----"
76 PRINT L+M
```

VAL stands for "value." It changes what is in the string to a number, if it can.



MAKING NUMBERS INTO STRINGS

Run:

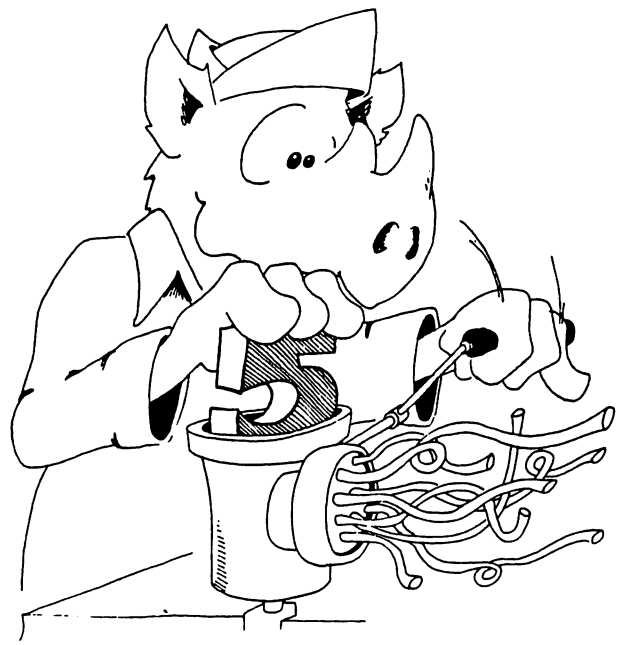
```
10 REM MAKING NUMBERS INTO STRINGS
11 REM
20 PRINT
25 INPUT"GIVE ME A NUMBER ":NB
30 N$=STR$(NB)
35 L=LEN(N$)
37 PRINT
40 FOR I=L TO 1 STEP -1
45 B$=B$ & MID$(N$,I,1)
50 NEXT I
60 PRINT"HERE IT IS BACKWARDS"
65 PRINT
66 PRINT B$
```

STR$ stands for "string." It changes a number into a string.

FUNCTIONS AGAIN

In this book we use these functions:

RND( )	INT( )	SEG$( )	LEN( )	POS( )
VAL( )	ASC( )	STR$( )	CHR$( )	



Rules about functions:

Functions always have ( ) with one or more "arguments" in them. Example:

SEG$(D$,5,J) has 3 arguments: D$, 5, and J

The arguments may be numbers or strings or both.

A "function" is not a "command." It cannot begin a statement.

right: `10 LET D=LEN$(CS$)`

wrong: `10 LEN (CS$)=5`

A function acts just like a number or a string. We say the function "returns a value." The value can be put into a box or printed just like any other number or string. The function may even be an argument in another function.

The arguments help pick which value is returned.

(Remember, string values go into string variable boxes, numeric values go into numeric boxes.)

PRACTICE WITH FUNCTIONS

For each function in the list below:

Tell how many arguments it has and give their names. Tell whether the value of the function is a string or a number:

INT(Q) .....

SEG$(R$,E,2) .....

VAL(ER$) .....

STR$(INT(RND*30)) .....

Each line below has errors. Explain what is wrong.

10 INT(Q)=65 .....

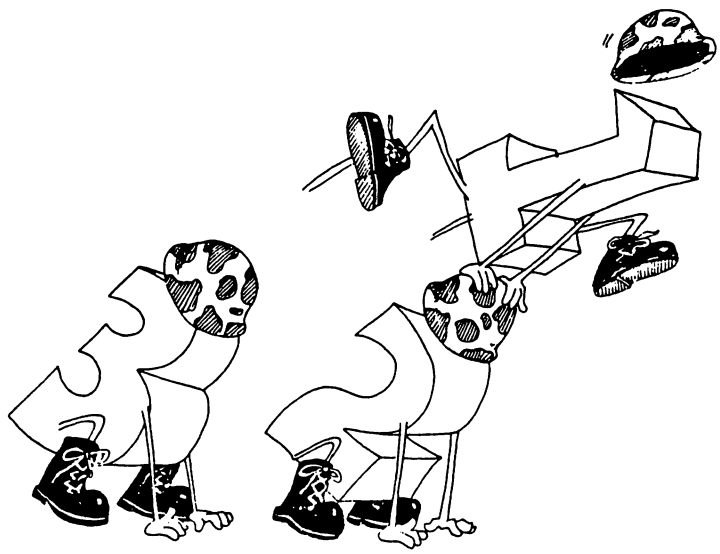
10 D$=SEG(R$,1) .....

10 PW$=VAL(F$) .....

10 PRINT CHR$ .....

Assignment 27:

1. Write a program which asks for a number. Then make another number which is backwards from the first, and add them together. Print all three numbers like an addition problem (with "+" sign and a line under the numbers).
2. Make a number "leapfrog" slowly across the screen. That is, write it on the screen, then take its left digit and put it on the right. Keep repeating. Don't forget to erase each digit when you move it.



INSTRUCTOR NOTES 28 JOYSTICKS FOR GAMES

TI calls its joysticks "Wired Remote Controllers." This lesson introduces the function CALL JOYST. The statement CALL KEY is used to test if the "fire button" is being pressed.

Joysticks are commonly used in animated graphics games. In this lesson, the joystick is used to move a dot around on the screen.

The JOYST(N,X,Y) statement has 3 arguments. The first indicates which stick is being interrogated. The second contains −4, 0, or 4 depending whether the stick is left, center or right. The third contains −4, 0, 4 depending on whether the stick is up, center or down.

The student will need to understand the X,Y addressing of the squares on the 32 by 24 screen.

When drawing moving objects, you need to erase each old image before the next image is drawn. The erasing is best done just before the new dot is drawn, to minimize flicker on the screen.

BASIC is very slow for action games. Maximum speed can be obtained if the "working" part of the program is first, and the "initialization" part is at the end, reached by a GOTO in line 1 or 2. This idea is further developed in the lesson on user friendly programs.

The joystick is also useful for picking selections in menus.

QUESTIONS:

1. How do you ask the joystick what direction it is being pushed?
2. How do you erase a dot before moving it?
3. How do you tell if the button on the joystick is being held down?

LESSON 28 JOYSTICKS FOR GAMES

Texas Instruments calls its joysticks "Wired Remote Controllers."

Plug the joystick cable into the socket on the left side of the computer.

(Do NOT plug it into the socket on the back of the computer. That is where the tape recorder plugs in.)

THE JOYSTICKS

There are two joysticks. Test them out with this program.

Run:

```
10 REM === JOYSTICKS ===
15 CALL CLEAR
30 PRINT "PUSH THE JOYSTICKS AROUND"
38 REM CHECK THE STICKS
44 PRINT" X Y H V"
45 FOR I=1 TO 23
50 CALL JOYST(1,X,Y)
51 CALL JOYST(2,H,V)
60 PRINT X;Y;H;V
65 NEXT I
70 PRINT
99 GOTO 44
```

Use the FCTN CLEAR keys to end the program. Save to tape.

THE CALL JOYST STATEMENT

Use the CALL JOYST(1,X,Y) statement to ask which way joystick 1 has been pushed.

Left, center, or right puts −4, 0, or 4 into the X variable box.

Up, center, or down puts 4, 0, or −4 into the Y variable box.

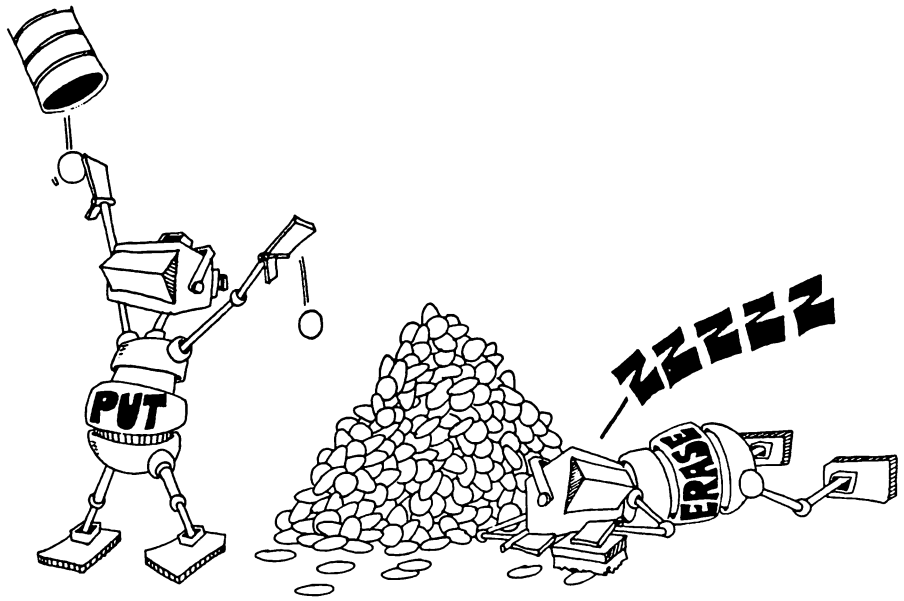
CALL JOYST(2,X,Y) does the same for stick 2.

Run:

```
10 REM MOVE A SPOT
12 CALL CLEAR
25 CALL CHAR(42,"FFFFFFFFFFFFFFFF")
27 CALL COLOR(2,7,1)
50 X=15
51 Y=12
59 REM ----------------LOOK AT JOYSTICK
60 CALL JOYST(1,H,V)
61 DX=H/4
62 DY=V/4
63 REM ----------------ERASE OLD SPOT
64 CALL HCHAR(Y,X,32)
67 X=X+DX
68 Y=Y-DY
79 REM ----------------PUT SPOT ON SCREEN
80 CALL HCHAR(Y,X,42)
99 GOTO 60
```

Use the FCTN CLEAR keys to stop the program. Save to tape.

In line 68, you need the negative sign because HCHAR measures Y down while the joy stick measures Y up.



ERASE AND PUT

"Erase and put, erase and put ... ." Every time you put a dot, you have to erase it again before putting it somewhere else. Otherwise, you will get more and more dots. (To see this happen, remove line 64.)

Line 64 erases the old dot.

Line 80 puts a new dot on the screen.

Assignment 28A:

1. Add a border to the MOVE A DOT program.
2. Make the dot stop when it moves up to any border.

THE FIRE BUTTON ON THE JOYSTICK

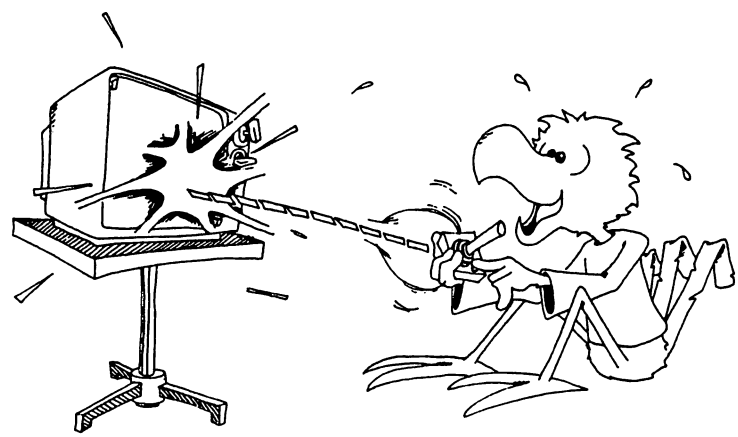
Use CALL KEY( ) to see if the fire button is down.

Add:
```
70 CALL COLOR(2,7,1)
85 CALL KEY(1,K,S)
86 IF S=0 THEN 99
89 CALL COLOR(2,8,1)
```

This CALL KEY statement is just like the one in Lesson 23. The status variable S is zero when you are pushing the fire button.

Assignment 28B:

1. Change the program above so that the dot disappears when the fire button is pushed.



INSTRUCTOR NOTES 29 LONG PROGRAMS

This lesson demonstrates top down organization of a task.

One of the hardest habits to form in some students (and even some professionals) is to impose structure on the program. Structuring has gone by many names such as "structured programming" and "top down programming" and uses various techniques to discipline the programmer.

Here we outline the program right on the screen. The task is "chunked" into sections by using subroutines. This leads to clarity in the articulation of the program parts and allows testing and debugging of each part separately from the others.

After the outline is done, each subroutine is expanded by writing in ordinary English what needs be done. Only when the English description is itself sufficiently detailed, does the BASIC programming begin.

Of course, there is always some backing and filling to be done as the program is written. The number of subroutines may change and the tasks performed in each will also change, usually expand.

There are those who advocate performing all planning of the program on paper before starting coding at all. This may work for some programmers, but children especially are unlikely to adopt this style of work. Besides, if one advocates word processors so that writing text can be done interactively on the screen, it would seem equally appropriate to plan computer programs on the screen.

QUESTIONS:

1. Why is it good to outline the program on the screen?
2. If you have trouble deciding what steps go in a game program, how can you, a friend and a piece of paper help?
3. What do you do (in English) to the outline next?
4. When do you test each subroutine that you have written?

LESSON 29 LONG PROGRAMS

HOW TO WRITE A LONG PROGRAM

Let's write a hangman game. This is a word guessing game where you draw another part of the hanging person each time you make a wrong guess for a letter.

First make an outline. You can do this on paper or right on the screen.

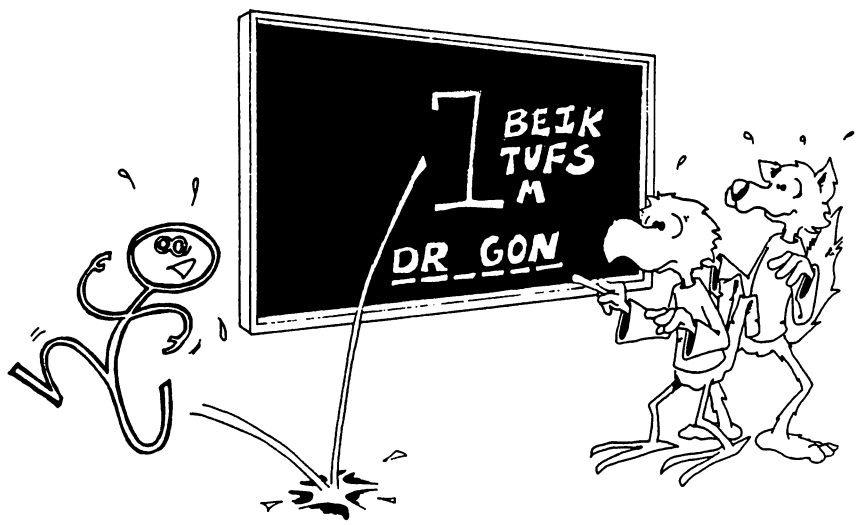
If you have trouble deciding what to do, then get a friend to play through a game with you. Keep careful track on paper of what is done during the game. Then the progam has to do the same things.

The outline could be:

```
10 REM *** HANGMAN GAME ***
200 REM INSTRUCTIONS
300 REM GET THE WORD TO GUESS
400 REM MAKE A GUESS
500 REM TEST IF RIGHT
600 REM ADD TO THE DRAWING
700 REM TEST IF GAME IS OVER
800 REM END GAME MESSAGE
```

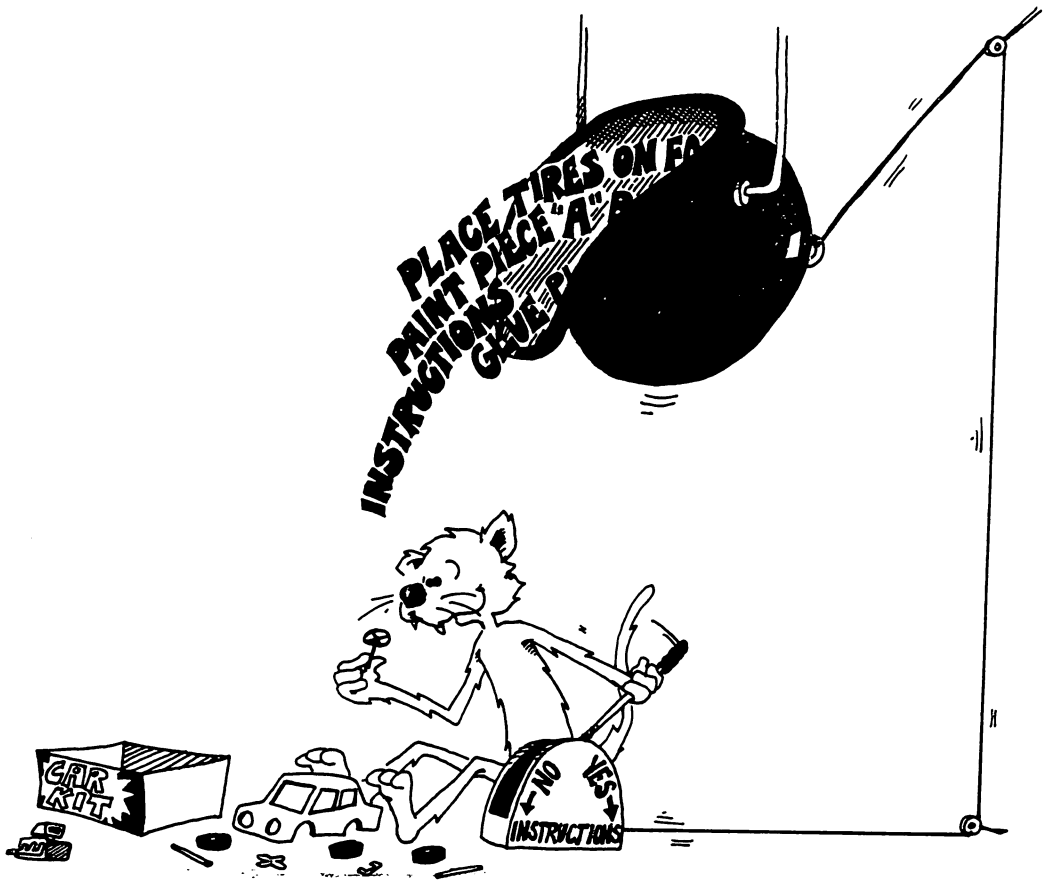
SAVE to tape.

After making this outline, fill in more details. Just write in Engish what each subroutine needs to do.



```
10 REM === HANGMAN GAME ===
99 REM
100 REM---------------------- MAIN LOOP
101 REM
120 INPUT" NEED INSTRUCTIONS? <Y/N> ": Y$
122 IF Y$="Y" THEN 200
130 REM---------------------- GET WORD
131 GOSUB 300
132 STOP
135 REM---------------------- MAKE GUESS
136 GOSUB 400
140 REM---------------------- TEST GUESS
141 GOSUB 500
145 REM---------------------- TEST IF GAME IS OVER
146 GOSUB 700
190 REM---------------------- MAKE ANOTHER GUESS
191 GOTO 135
200 REM INSTRUCTIONS
```

--- write the instructions last



```
290 GOTO 130
299 REM
300 REM---------------------- GET THE WORD TO GUESS
301 REM
```

--- use INPUT to get a word from player 1
--- draw dashes for the letters to be guessed

```
390 RETURN
399 REM
400 REM---------------------- MAKE A GUESS
401 REM
```

--- player 2 guesses a letter

```
490 RETURN
499 REM
500 REM---------------------- TEST IF GUESS IS RIGHT
501 REM
```

--- if wrong, GOSUB 600, draw hangman part
--- if right, GOSUB 700, see if game is over

```
590 RETURN
599 REM
600 REM---------------------- ADD TO THE DRAWING
601 REM
```

--- add to the hangman drawing
--- test if drawing is done
--- if so, then GOSUB 800

```
690 RETURN
699 REM
700 REM---------------------- TEST IF GAME IS OVER
701 REM
```

--- see if all letters have been guessed
--- if yes, GOSUB 900

```
790 RETURN
799 REM
800 REM---------------------- END GAME MESSAGE
801 REM
```

--- message for when guesser loses

```
890 RETURN
899 REM
900 REM---------------------- END GAME MESSAGE
901 REM
```

--- message for when guesser wins

```
990 RETURN
```

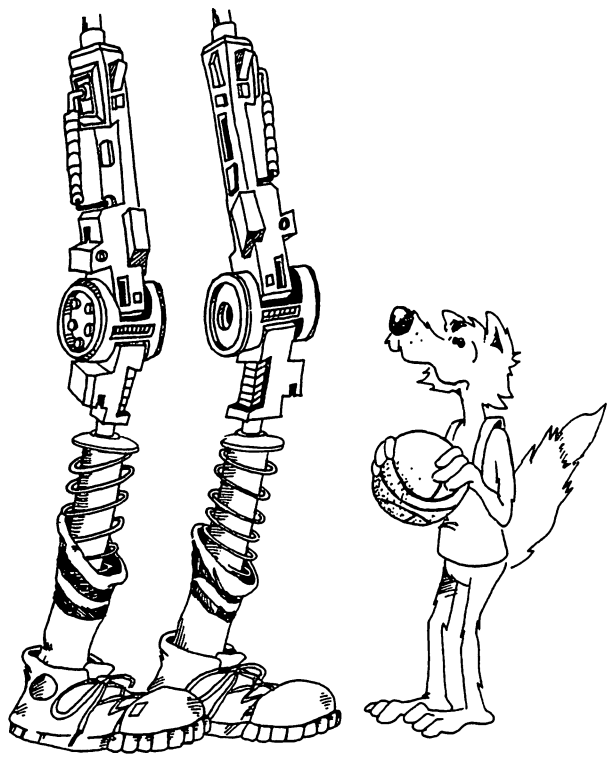
SAVE to tape

Now is the time to start writing and testing the first part of the program. Put a STOP in line 132 so that only the first subroutine will be run. (After the first subroutine works OK, take the STOP out. See Lesson 33.)

Start by writing the subroutine at 300, GET A WORD. The first step is to write more details, in English, of what the subroutine needs to do. Then start writing the BASIC lines.

Assignment 29:

1. Finish the hangman game. This is a long project. Start by writing the GET A WORD subroutine. Then SAVE it to tape. You may want to write one subroutine each day until the program is done.



INSTRUCTOR NOTES 30 ARRAYS AND THE DIM STATEMENT

This lesson introduces arrays. The DIM( ) statement is described.

TI BASIC allows arrays with one, two, or three indices.

Arrays with one index are described first. The array itself is compared to a family, and the individual elements of the array to family members, with the index value being the "first name" of the member.

Two dimensional arrays are compared to the arrangement of numbers on a calender month page or the rectangular array of cells on the TV screen.

Three dimensional arrays are just mentioned, with no examples given.

Arrays themselves are not too difficult a concept. The trick is to see how they help in programming. There are a large variety of uses of arrays, and many do not seem to fall into recognizable catagories.

One can use them to store lists of information. Connected lists also can occur. The telephone number program uses two linear arrays: one for names, the other for numbers. They are indexed the same, so a single index number can retrive both the name and the number that goes with it.

Another general use of matrices is to store numbers which cannot be neatly obtained from an equation. An example would be the length in days of the 12 months.

Games often use arrays to store information about the playing board.

QUESTIONS:

1. What does the DIM AD$(5) command do?
2. Where do you put the DIM command in the program?
3. What two kinds of array families are there?
4. What is the "index" or "subscript" of an array?
5. What does the command DIM SR(5,9) do?

LESSON 30 ARRAYS AND THE DIM STATEMENT

MEET THE ARRAY FAMILY

```
22 F$(0)="DAD"
24 F$(1)="MOM"
26 F$(2)="MINDA"
```

Each member of the family is a variable. The F$ family are string variables.

Here is a family of numeric variables:

```
35 N(0)= 43
37 N(1)= 13
39 N(3)= 0
41 N(4)= 0
```

The family has a "last name" like A( ) or B$( ). Each member has a number in ( ) for a "first name." The array always starts with the first name "0".

Instead of "family" we should say "array."

Instead of "first name" we should say "index number" or "subscript."

THE DIM( ) COMMAND SAVES BOXES

When the array family goes to a movie, they always reserve seats first. They use a DIM command to do this.

The DIM ... command tells the computer to reserve a row of boxes for the array. DIM stands for "dimension" which means "size."

For example, the statement

```
18 DIM A(3)
```

saves four memory boxes, one each for the variables A(0), A(1), A(2) and A(3). These boxes are for numbers and contain the number "0" to start with. Another example:

```
30 DIM A(3),B$(4)
```

This time, DIM reserves 4 boxes for the A( ) array and 5 for the string array B$( ). The boxes named B$(0) through B$(4) are for strings and are empty to start with.

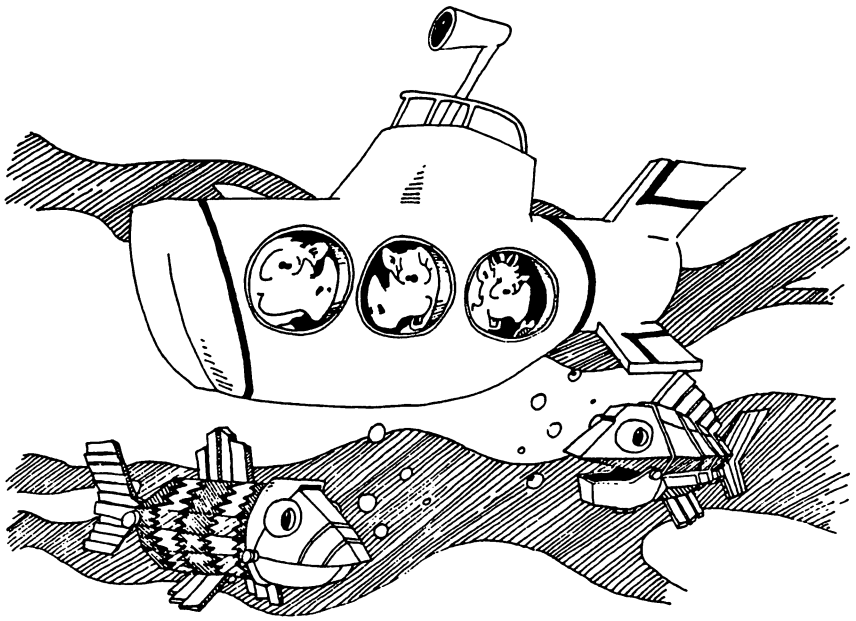
Rule: Put the DIM( ) statement early in the program, before the array is used in any other statement.

MAKING A LIST

Enter:

```
10 REM +++ IN A ROW +++
20 CALL CLEAR
30 DIM A$(5)
35 PRINT"ENTER A WORD"
40 FOR N=1 TO 5
50 INPUT A$(N)
55 PRINT
57 IF N=5 THEN 60
59 PRINT "ANOTHER"
60 NEXT N
70 PRINT
100 REM PRINT THEM
105 PRINT"HERE THEY ARE ALL MIXED UP"
106 I=INT(RND*5) + 1
120 PRINT A$(I);" ";
130 GOTO 106
```

Run and save to tape.



You can use a member of the array by itself; look at this line:

```
40 B$(2)="YELLOW SUBMARINE"
```

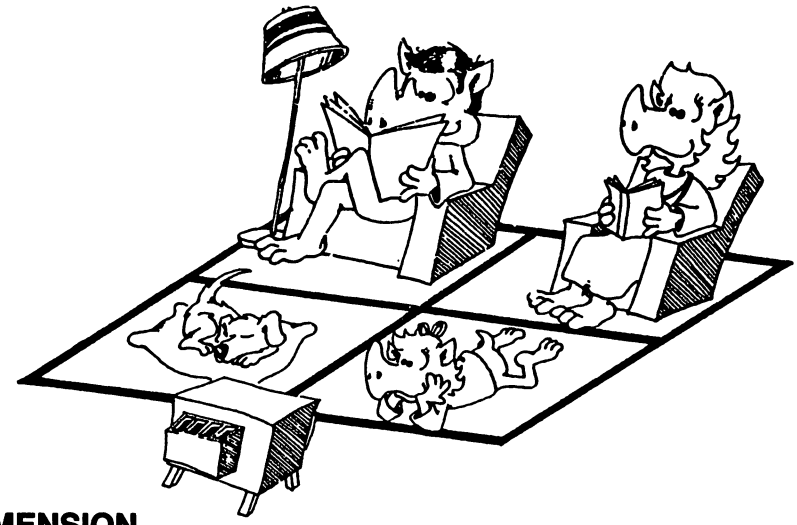
Or the array can be used in a loop where the index keeps changing. Lines 50 and 120 in the program "IN A ROW" do this.

MAKING TWO LISTS

Enter:

```
10 REM PHONE LIST
20 CALL CLEAR
30 DIM NA $(20), NU $(20)
35 I=0
40 PRINT "ENTER NAMES AND NUMBERS "
50 PRINT
51 INPUT "NAME? ":NA$(I)
60 INPUT "NUMBER? ":NU$
70 I=I+1
71 GOTO 50
```

Run. Press the FCTN CLEAR keys to stop program. Save to tape.



ONE DIMENSION, TWO DIMENSION, ...

The arrays which have one index are called one dimensional arrays. But arrays can have 2 or 3 indices. Two dimensional arrays have their "family members" put into a rectangle like the days in a month on a calendar.

```
10 REM +++ TWO-DIM ARRAY +++
15 REM
18 CALL CLEAR
20 DIM T(5,6)
30 FOR X=0 TO 5
40 FOR Y=0 TO 6
50 T(X,Y)= X+Y
60 NEXT Y
61 NEXT X
65 REM
70 REM ------ PRINT OUT THE ARRAY
72 REM
80 FOR J=0 TO 6
82 PRINT
85 FOR I=0 TO 5
88 PRINT TAB(I*3);T(I,J);
90 NEXT I
91 NEXT J
```

Assignment 30:

1. Write a program which stores the number of days in each month in an array. Then when you ask the user to enter a number <1 to 12>, it prints out the number of days in that month.
2. Finish the PHONE LIST program so that it prints out the list of names with the telephone numbers beside them.
3. Use a two dimensional array to make a "weekly school calendar" program. It could use an array made by DIM AR$(5,6) so that each day of the week could have an entry for each class hour.

INSTRUCTOR NOTES 31 LOGIC: TRUE AND FALSE

This lesson treats the numeric values for TRUE and FALSE. The student is encouraged to use homemade DO WHILE and DO UNTIL constructions.

There are two abstract ideas in this lesson which may give difficulty.

One is that TRUE and FALSE have numeric values of -1 and 0. Any expression which is of the form of an assertion (a "phrase A"), has a numeric value of 0 or -1. This number can be treated just as any other number. It can be stored in a numeric variable, printed, or used in an expression. Most often it is used in an IF statement.

The other abstract idea compounds the confusion. The IF command doesn't really look to see if "phrase A" is present. Rather, it looks for a numeric value between IF and THEN. Any number which is non-zero is treated as TRUE.

(The value -1 for TRUE may seem artificial. It comes from the idea that FALSE is zero, which is a natural idea. Then NOT FALSE should be TRUE. But the Boolean operation NOT on a number whose digits are all zero gives a number whose digits are all ones. This number has the value -1 in the usual two's complement notation used on the binary level of the computer's operation.)

It is easy to build a program using GOTO and IF branching that ends up being a bowl of spaghetti. The programmer must discipline herself to avoid this. A universal scheme limits the programs to single-entry, single-exit blocks. Then those which make conditional branches (IFs) are limited to just two:

DO UNTIL and DO WHILE.

The difference is whether the test for exit is made at the beginning or at the end of the block.

QUESTIONS:

1. What do you call this block? (DO UNTIL or DO WHILE)

```
20 I=1
30 IF I>9 THEN 60
40 I=I+1
45 PRINT I
50 GOTO 30
60 REM
```

2. Write your own DO UNTIL block.
3. Write your own DO WHILE block.
4. What number will each of these lines print?

```
10 PRINT 3<>4
10 PRINT 8>9
```

LESSON 31 LOGIC: TRUE AND FALSE

TRUE AND FALSE ARE NUMBERS

The computer says true and false are numbers.

Rule: TRUE is the number −1
FALSE is the number 0

(It is easy to remember that 0 is FALSE because zero is the grade you get if your homework is false.)

To see these numbers, enter these commands:

```
PRINT 3=7    prints 0     because    3=7 is FALSE
PRINT 3=3    prints -1    because    3=3 is TRUE
```

PUTTING TRUE AND FALSE IN BOXES

The numbers for TRUE and FALSE can be put into boxes just like other variables:

Run:
```
10 N = (3=22)
20 PRINT N
```

The number 0 is stored in the box N because 3=22 is FALSE.

Run:
```
10 N = "B"="B"
20 PRINT N
```

The number −1 is stored in the box N because the two letters in the quotes are the same, so the statement "B"="B" is TRUE.

THE IF COMMAND TELLS LITTLE WHITE LIES

The IF command looks like this:

```
10 IF phrase A THEN line number N
```

Try this:

```
10 IF 0 THEN 30
20 PRINT "FALSE"
21 END
30 PRINT "TRUE"
```

It should print `FALSE`.

Try it again with this line:

```
10 IF -1 THEN 30
```

It should print `TRUE`.

Now with this:

```
10 IF 22 THEN 30
```

What does it print? .....

Rule: In an IF, the computer looks at "phrase A":

If it is zero, the computer says "phrase A is FALSE", and skips what is after THEN.

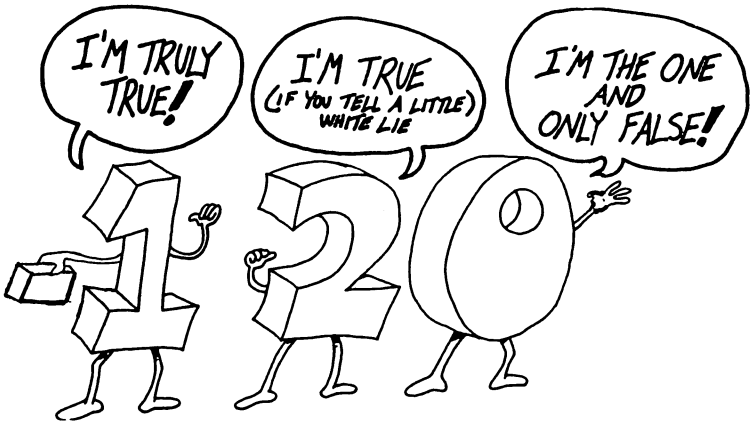
If it is not zero, the computer says "phrase A is TRUE", and goes to the line whose number is after THEN.

So the IF command tells little white lies:

TRUE is supposed to be the number "−1".

But the "IF" stretches the truth to say "TRUE is anything that is not FALSE."

That is, any number that is not zero is TRUE.



THE LOGIC SIGNS

You can use these 6 symbols in the "phrase A":

=	equal
<>	not equal
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal

You have to press the SHIFT key and two other keys to make the <> sign and the <= and >= signs.

The last two are new so look at this example to see the difference between < and <= :

2<=3	is	TRUE	2<3	is	TRUE
3<=3	is	TRUE	3<3	is	FALSE
4<=3	is	FALSE	4<3	is	FALSE

Assignment 31A:

1. Tell what will be found in the box N if:

```
N = 4=4
N = "G"< >"S"
N = 5>7
N = 5>=4
```

2. Tell if the word "JELLYBEAN" will be printed:

```
IF Ø          THEN PRINT "JELLYBEAN"
IF 1          THEN PRINT "JELLYBEAN"
IF 9          THEN PRINT "JELLYBEAN"
IF 3< >Ø      THEN PRINT "JELLYBEAN"
IF "A"="Z"    THEN PRINT "JELLYBEAN"
IF 4<=5       THEN PRINT "JELLYBEAN"
```

ONE DOOR TO GO IN AND ONE DOOR TO GO OUT

Use your IF and your GOTO statements carefully to avoid "spaghetti" programs.

Each IF should be used in a block of code which has one entrance and one exit.

There are only two kinds of blocks you need for programming any IF idea:

DO UNTIL ...

```
58 ---------
60 REM DOOR IN
62 ---------
64 ---------
66 ---------
68 IF A THEN 60
70 REM DOOR OUT
72 ---------
```

The lines in the DO UNTIL block keep repeating until phrase A is false.

DO WHILE ...

```
58 ---------
60 REM DOOR IN
62 IF A THEN 70
64 ---------
66 ---------
68 GOTO 62
70 REM DOOR OUT
72 ---------
```

The lines in the DO WHILE block keep repeating while phrase A is true.

Assignment 31B:

1. Go back and look at the examples is this book. Find "IF" blocks which are like DO UNTIL ... . Find "IF" blocks which are like DO WHILE ... . Find ones that are "spaghetti"!
2. Write a program to detect a double negative in a sentence. Look for negative words like not, no, don't, won't, can't, nothing, and count them. If there are 2 such words there is a double negative. Test the program on the sentence "COMPUTERS AIN'T GOT NO BRAINS".

INSTRUCTOR NOTES 32 USER FRIENDLY PROGRAMS

This lesson concerns clear programs which interact with the user in a "friendly" way.

The "spaghetti" program should be discouraged. A format for writing programs is presented in this lesson. While methods of imposing order on the task are largely a matter of taste, the methods used in this lesson can serve to introduce the ideas.

"User friendly" means that the screen displays are easy to read, keyboard input is "ENTER key free" as much as possible, and errors are "trapped." Ask if entries are OK. If not, give an opportunity to fix things.

Instructions and "HELP" should be available. Prompts need be given. Beginners need complete prompts, but experienced users would rather have curt prompts.

It is hard to teach the writing of "user friendly" programs. Success depends mostly on the attitude of the programmer. The best advice is to "turn up your annoyance detectors to high" as you write and debug the program.

Most young students will not progress very far toward fully "friendly" programming. To be acquainted with the desirability of "friendly" programming and to use some simple techniques toward accomplishing it are satisfactory achievements.

QUESTIONS:

1. Should your program give instructions whether the user wants them or not?
2. What is a "prompt"? Give two examples.
3. If you want the user to enter a single letter from the keyboard, what command is best? (Avoids using the ENTER key.)
4. What is an "error trap"? How would you trap errors if you asked your user to enter a number from 1 to 5?
5. In what part of the program are most of the GOSUB commands found?
6. Why put the "STARTING STUFF" section of the program at the end of the program (at high line numbers)?

LESSON 32 USER FRIENDLY PROGRAMS

There are two kinds of users:

1. Most want to run the program. They need:

    instructions
    prompts
    clear writing on the screen
    no clutter on the screen
    erasing old stuff from the screen
    not too much key pressing
    protection from their own stupid errors

2. Some want to change the program. They need:

    a program made in parts
    each part with a title in a REM
    explanations in the program

(Don't forget you are a user of your own programs, too! Be kind to yourself!)

PROGRAMS HAVE THREE PARTS

"STARTING STUFF": at the beginning of the program run.

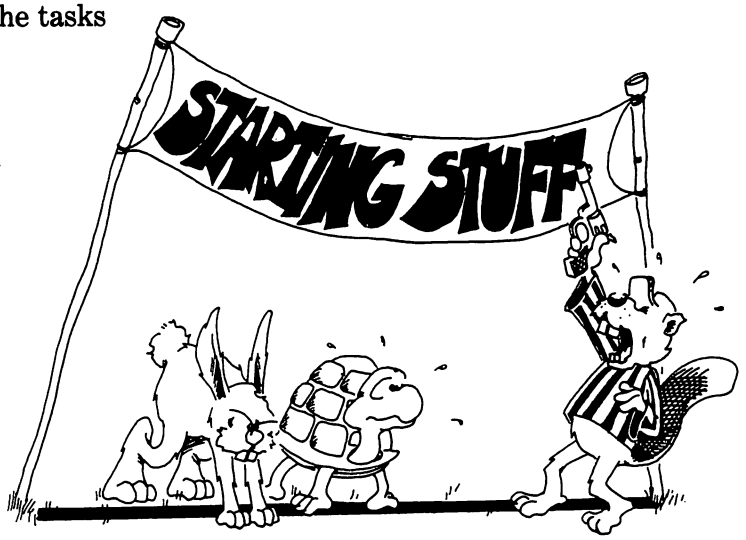
    give instructions to the user
    draw a screen display
    set variables to their starting values
    ask the user for starting information

MAIN LOOP:

    controls the order in which tasks are done
    calls subroutines to do the tasks

SUBROUTINES:

    do parts of the program



PROGRAM OUTLINE

```
  1 REM       program name ***
  2 GOTO 1000
---
100 REM MAIN LOOP
---
---       calls subroutines
---
199 END
1000 REM
1001 REM        program name ***
1002 REM
---- REM  REM's that give a description of the
----       program, variable names, etc.
----
1999 REM
2000 REM STARTING STUFF
----
----       ask for starting information
----       set variable values
----       give instructions
----
2999 GOTO 100
```

PUT THE MAIN LOOP AT THE BEGINNING OF THE PROGRAM

Put the MAIN LOOP near the front because it will run faster there.

PUT STARTING STUFF AT THE END OF THE PROGRAM

Put the STARTING STUFF near the back because it may be the biggest part of the program, and you may keep adding to it as you write, to make the program more "user friendly." It does not need to run fast.

PUT SUBROUTINES IN THREE PLACES

between line 2 and line 99 for subroutines that must run fast
after line 2999 for starting stuff subroutines
between line 200 and 999 for the rest of the subroutines

INFORMATION PLEASE

```
380 PRINT "DO YOU WANT INSTRUCTIONS? <Y/N> "
```

This lets a beginner see instructions and lets others say "no."

TIE A STRING AROUND THE USER'S FINGER

Use a "prompt" to remind users what choices they have.

Example: <Y/N> where the choice is Y for "yes" or N for "no".

Beginners need long prompts. Other users like short prompts.

OUCH! MY FINGERS HURT

Use the CALL KEY command to enter single letters. This saves having to press ENTER.

```
380 PRINT "DO YOU WANT INSTRUCTIONS? <Y/N> "
381 CALL KEY(0,K,S)
382 IF K=-1 THEN 381
383 IF CHR$(K)="Y" THEN 900
```

SET TRAPS FOR ERRORS

Example: Add this line to the above lines:

```
384 IF CHR$(K)<>"N" THEN 380
```

Line 380 asked for only two choices, Y or N. If the user presses some other key, line 384 sends him back to line 380.

Traps make your program "bomb proof" so that users will be unable to goof it up!

Assignment 32:

1. Look at the COLOR EATER program. Add REM's to explain the lines in the program. Fix up the program to be user friendly.
2. Write a secret cipher program. The user chooses a password and it is used to make a cipher alphabet like this:

   if the password is DRAGONETTE
   remove the repeated letters, get DRAGONET
   put it at the front of the alphabet and the rest of
   the letters after it in normal order

   DRAGONETBCFHIJKLMPQSUVWXYZ

The user chooses to code or decode from a menu.

```
2 REM *** COLOR EATER ***
3 GOTO 1000
100 REM --------------------MAIN LOOP
101 REM
110 CALL HCHAR(Y,X,32)
115 X=X+INT(RND*3)-1
116 Y=Y+INT(RND*3)-1
120 IF X<>0 THEN 122
121 X=1
122 IF X<>33 THEN 124
123 X=32
124 IF Y<>0 THEN 126
125 Y=1
126 IF Y<>25 THEN 130
127 Y=24
130 CALL GCHAR(Y,X,N)
133 IF N=32 THEN 140
136 CALL SOUND(50,900,10)
140 CALL HCHAR(Y,X,50)
150 A=INT(RND*32)+1
151 B=INT(RND*24)+1
155 CALL HCHAR(B,A,58)
199 GOTO 100
1000 REM
1001 REM === STARTING STUFF ===
1002 REM
2000 CALL CLEAR
2002 RANDOMIZE
2005 F$="FFFFFFFFFFFFFFFF"
2010 CALL CHAR(42,F$)
2012 CALL CHAR(50,F$)
2013 CALL CHAR(58,F$)
2020 CALL COLOR(2,9,5)
2021 CALL COLOR(4,5,1)
2025 CALL SCREEN(15)
2100 FOR I=1 TO 300
2120 X=INT(RND*32)+1
2121 Y=INT(RND*24)+1
2140 CALL HCHAR(Y,X,42)
2190 NEXT I
2200 REM -------------------EATER POSITION
2210 X=16
2220 Y=12
2999 GOTO 100
```

INSTRUCTOR NOTES 33 DEBUGGING, STOP, CON

If we had NASA's budget and time scale, we might teach debugging by systematically categorizing all types of errors, preparing a set of "bad" programs containing these errors, and conducting "lab" sessions to drill in debugging techniques.

The "sigh and moan" technique being a loser, our students need a bag of tricks which help isolate program bugs, and should practice on programs that they are writing as they go through this book.

The inexperienced debugger feels hopeless inertia when "it doesn't work right." Rather than sit and stare, it is more useful to try some changes. Any changes are better than none, but random changes are very inefficient. The best changes are those which eliminate sections of the program from the list of possible hiding places for the bug.

As programs grow in complexity, more of the bugs result from unforeseen interactions between separated parts of the program. The bag of tricks we offer helps find these also. Delay loops, PRINT commands, and STOP statements help the student see how the program is functioning.

Many people overlook those techniques they can use after the program is stopped with a FCTN CLEAR, STOP, or an END. You can PRINT out any variable values you like so as to see what the program has done. You can also do arithmetic in the PRINT command to check what the program should be doing.

QUESTIONS:

1. How can you make the computer print

   ```
   BREAKPOINT AT 55
   ```

   by adding a line in the program?
2. Are the STOP and the END commands different?
3. How are the STOP and FCTN CLEAR commands different?
4. What does the CON command do?
5. Why would you put STOP commands into your program?
6. How do delay loops help you debug a program?
7. How do extra PRINT commands help you debug a program?
8. Why do you take the STOP and extra PRINT commands out of the program after you have fixed the errors?
9. Can you pick in what line the FCTN CLEAR keys will stop the program? Can you pick using the STOP command?

LESSON 33 DEBUGGING, STOP, CON

THE STOP COMMAND

Enter and run:

```
10 REM SECRET STOP
20 CALL CLEAR
25 G = INT(RND*200)
30 FOR I=0 TO 200
40 IF I<>G THEN 50
45 STOP
50 NEXT I
```

The program will stop.

What do you suppose the secret value of I was?

Enter: `PRINT I` (No line number)

and find out.

"STOP" is just like "END". It doesn't matter which you use.

STOP makes the computer stop and enter the command mode.

You can have as many STOP commands in your program as you like.

STOP is used for debugging your program.

ANOTHER WAY TO STOP RUNNING THE PROGRAM

You can stop running the program with FCTN CLEAR. This means you hold down the key that says FCTN on it, and then press the "4" key.

Try it:

```
10 REM GO FOREVER
15 PRINT
20 PRINT "MUD TURTLES OF THE WORLD"
30 PRINT "UNITE!"
35 PRINT
40 FOR T=1 TO 400
41 NEXT T
99 GOTO 10
```

The command FCTN CLEAR stops the program wherever it is. It prints:

`BREAKPOINT AT XX` peeps and enters the command mode

(XX is the line number where it stops.)

The command CON starts the program again at the same spot.

WHAT DO YOU DO AFTER YOU STOP?

You put STOP in whatever part of your program is not working right. Then you run the program. After it stops, you look to see what happened.

(Or you use FCTN CLEAR to stop the program, but it may not stop in the spot where the trouble is.)

Put on your thinking cap. Ask yourself questions about what happened as the program ran.

You are in the command mode. You can:

List parts of the program and study them.

Use the PRINT command to look at variables. Do they have the values you expected?

Use LET to change values of variables.

If you find the trouble, you may add lines, change lines, or delete lines.

STARTING THE PROGRAM AGAIN

There are three ways to start a program. They are:

CON	for CONTINUE
RUN XX	where XX is a line number
RUN	your old friend

You may use the CON command if you stopped the program with FCTN CLEAR and did not make any changes in the program.

Otherwise, your only choice is to use RUN.

You can start at the lowest line number using RUN,

or you can start at any line XX using RUN XX.

Either way, RUN or RUN XX, the variable boxes are emptied before the program is run.

DEBUGGING

Little errors in your program are called "bugs."

If your program doesn't run right, do these four things:

1. If the computer printed an ERROR MESSAGE, it tells what line it stopped on. Careful, the mistake may really be in another line!
2. If the computer just keeps running but doesn't do the right thing, stop it and put in some PRINT lines which will tell what is happening.
3. Or you can put STOP commands into the program.
4. If the program runs so fast that you can't tell what is happening, put in some delay loops to slow it down.

After you have fixed the program, take the PRINT lines, the STOPs and the delay loops out of the program.

Assignment 33:

1. Go back to the SNAKE program and fix up some of the bugs. For example, the program "crashes" when the snake hits a wall. Add "food" for the snake. Add score keeping. Let the game end if the snake touches a wall.
2. Go back and fix up some other program that you have written.


DON'T
FORGET...

ANSWERS TO ASSIGNMENTS

1-2

```
10 REM GREETING
20 PRINT "HI THERE,"
30 PRINT "TI COMPUTER"
```

2-1

```
10 REM NAMES
20 PRINT "MINDA"
30 PRINT "ANNE"
40 PRINT "CARLSON"
```

2-3

```
10 REM NAMES
20 CALL SOUND(300,400,10)
22 CALL SCREEN(5)
25 PRINT " MINDA"
30 CALL SOUND(300,600,10)
32 CALL SCREEN(10)
35 PRINT " ANNE"
40 CALL SOUND(500,800,10)
42 CALL SCREEN(15)
45 PRINT " CARLSON"
```

3-6

```
10 REM BIRDS
15 CALL CLEAR
20 PRINT
22 CALL SOUND(100,1000,10)
25 PRINT " ---O---"
30 PRINT
40 PRINT
42 CALL SOUND(150,900,10)
50 PRINT "          ---O---"
60 PRINT
70 PRINT
75 CALL SOUND(100,1100,15)
80 PRINT "      --O--"
```

4-2

```
10 REM SMILE
12 CALL CLEAR
20 PRINT
30 PRINT
40 PRINT
50 PRINT "    OO    OO        "
60 PRINT
61 PRINT
62 PRINT
63 PRINT " *                * "
64 PRINT "   *            *   "
65 PRINT "    *          *    "
66 PRINT "      ******      "
```

5-1

```
10 REM TALKING
15 CALL CLEAR
20 PRINT
22 PRINT
24 PRINT
30 PRINT "HELLO. WHAT IS YOUR NAME?"
32 PRINT
34 INPUT N$
36 PRINT
40 PRINT "WELL,"
42 PRINT
44 PRINT N$
46 PRINT
50 PRINT "IT IS SILLY TO TALK TO COMPUTERS!"
```

5-2

```
10 REM THE STRING BOX
12 CALL CLEAR
20 PRINT "WHAT IS YOUR FAVORITE COLOR?"
25 INPUT C$
27 PRINT
30 PRINT "I PUT THAT IN BOX C$"
32 PRINT
35 PRINT "NOW, YOUR FAVORITE ANIMAL?"
40 INPUT C$
42 PRINT
45 PRINT "I PUT THAT IN BOX C$ TOO"
47 PRINT
50 PRINT "NOW LET'S PRINT WHAT IS IN BOX C$"
```

```
52 PRINT
55 PRINT "IT IS:"
57 PRINT
60 PRINT C$
```

6-2

```
10 REM MUSIC
12 CALL CLEAR
20 PRINT "WHAT IS YOUR FAVORITE MUSICAL GROUP?"
25 INPUT G$
27 CALL CLEAR
30 PRINT "WHAT TUNE DO THEY PLAY?"
35 INPUT T$
40 CALL CLEAR
50 PRINT
55 PRINT G$;" PLAYS ";T$
```

6-3

```
10 REM SAME AS ABOVE EXCEPT:
55 PRINT G$;
56 PRINT " PLAYS ";
57 PRINT T$
```

7-2

```
10 REM FEELINGS
15 CALL CLEAR
20 PRINT
22 PRINT
24 PRINT "HOW IS THE WEATHER?"
26 PRINT
28 INPUT W$
30 PRINT "AND HOW DO YOU FEEL?"
32 PRINT
34 INPUT F$
36 PRINT
38 PRINT "YOU MEAN:"
40 PRINT
45 S$ = W$ & " AND " & F$
50 PRINT S$
```

8-2

```
10 REM TEEN TIMES
11 REM
20 PRINT "TEEN POWER"
21 PRINT
22 PRINT
23 PRINT
30 GOTO 20
```

8-4

```
10 REM FRIENDS
15 CALL CLEAR
20 PRINT "MINDA"
22 CALL SOUND(200,200,10)
25 PRINT
30 PRINT "NELL"
40 CALL SOUND(200,400,10)
99 GOTO 20
```

9A-1

```
10 REM NICKNAMES
15 CALL CLEAR
20 PRINT"WHAT IS YOUR LAST NAME?"
22 PRINT
24 INPUT L$
28 CALL CLEAR
30 PRINT"SOMEONE TYPE THE NICKNAME"
32 PRINT
34 INPUT N$
36 CALL CLEAR
38 PRINT TAB(5);L$;TAB(15);N$
40 FOR T=1 TO 500
41 NEXT T
50 GOTO 10
```

9A-2

```
10 REM &%$%! INSULTS !%$%&
15 CALL CLEAR
16 PRINT
17 PRINT
```

```
20 PRINT "HEY YOU!! WHAT IS YOUR NAME?"
22 PRINT
25 INPUT N$
30 CALL CLEAR
31 PRINT
32 PRINT
35 PRINT N$
36 PRINT
37 PRINT
38 FOR T=1 TO 300
31 NEXT T
40 PRINT "BAH!!"
41 PRINT
42 PRINT
45 CALL SOUND(300,110,0)
50 PRINT "YOUR FATHER EATS LEEKS!!!"
```

10-1

```
10 REM BIRTH YEAR
15 CALL CLEAR
30 PRINT "HOW OLD ARE YOU?"
32 PRINT
34 INPUT A
36 PRINT
40 PRINT "AND WHAT YEAR IS IT NOW?"
42 PRINT
45 INPUT Y
50 B=Y-A
52 PRINT
55 PRINT "HAS YOUR BIRTHDAY COME YET THIS YEAR?"
58 PRINT "<Y/N>"
59 PRINT
60 INPUT Y$
65 IF Y$="N" THEN 70
66 REM SEE LESSON 13 FOR "IF"
67 B=B-1
70 PRINT
75 PRINT "YOU WERE BORN IN ";B;","
```

10-2

```
10 REM MULTIPLICATION
15 CALL CLEAR
20 PRINT
22 PRINT
```

```
24 PRINT
30 PRINT "GIVE ME A NUMBER"
32 PRINT
35 INPUT A
37 PRINT
38 PRINT
40 PRINT "GIVE ME ANOTHER "
42 PRINT
45 INPUT B
48 C=A*B
50 PRINT
52 PRINT
60 PRINT "THEIR PRODUCT IS ";C
```

11A-1

```
10 REM COUNTING BY FIVES
12 CALL CLEAR
20 FOR I=5 TO 100 STEP 5
30 PRINT I
35 FOR T=1 TO 100
36 NEXT T
40 NEXT I
```

11B-2

```
10 REM YOUR NAME IS CLIMBING
20 CALL CLEAR
25 PRINT "YOUR NAME?"
27 PRINT
30 INPUT N$
33 CALL CLEAR
35 FOR I = 1 TO 13
40 PRINT TAB(2*I);N$
45 NEXT I
```

11B-3

```
10 REM FRIENDS
15 CALL CLEAR
20 PRINT "GIVE ME YOUR NAMES"
25 INPUT N$,F$
30 FOR I = 1 TO 5
35 PRINT N$
36 PRINT F$
```

```
38 PRINT
40 FOR T=1 TO 300
41 NEXT T
50 NEXT I
```

11B-4

```
10 REM LOOPY TUNES
20 FOR I=1 TO 3
25 PRINT "SING"
26 PRINT
27 PRINT
30 FOR J=1 TO 3
34 CALL SOUND(300,J*200,10)
35 PRINT "TRA"
36 PRINT
40 FOR K=1 TO 3
44 CALL SOUND(200,J*200+50*K,10)
45 PRINT "LA ";
50 NEXT K
51 PRINT
52 PRINT
55 NEXT J
56 PRINT
57 PRINT
60 NEXT I
```

12-1

```
10 REM ** A PAIR OF DICE **
15 CALL CLEAR
20 LET D1=1+INT(RND*6)
22 LET D2=1+INT(RND*6)
25 D=D1+D2
30 PRINT "THE ROLL GAVE:"
32 PRINT
33 PRINT " THE FIRST DIE ";D1
34 PRINT " THE SECOND ";D2
35 PRINT " THE DICE ";D
47 PRINT
48 PRINT
50 PRINT "AGAIN?"
51 PRINT
55 INPUT Y$
60 IF Y$="Y" THEN 15
```

12-2

```
10 REM-------------- PAPER, SCISSORS, ROCK
12 CALL CLEAR
13 PRINT
14 PRINT
16 PRINT "PLAY THE "
18 PRINT
19 PRINT "   P A P E R "
20 PRINT "         S C I S S O R S "
21 PRINT "              R O C K "
22 PRINT
23 PRINT "GAME AGAINST THE COMPUTER"
24 PRINT
25 PRINT "PRESS 'CLEAR' KEY TO END GAME "
26 PRINT
27 PRINT "ENTER YOUR CHOICE <P,S,R>"
29 REM-------------- COMPUTER CHOOSES ITS MOVE
30 C=INT(RND*3)+1
31 C$="P"
32 IF C=1 THEN 36
33 C$="S"
34 IF C=2 THEN 36
35 C$="R"
36 REM-------------- C$ IS THE COMPUTER'S CHOICE
37 INPUT Y$
38 REM-------------- Y$ IS YOUR CHOICE
39 REM
40 REM-------------- IS THERE A TIE?
41 REM
50 IF C$<>Y$ THEN 60
52 REM-------------- THERE IS A TIE
55 PRINT " TIE"
57 GOTO 30
59 REM
60 REM-------------- NO TIE, WHO WINS?
61 REM
62 IF C$<>"P" THEN 70
63 IF Y$= "S" THEN 90
64 GOTO 80
70 IF C$<>"S" THEN 76
72 IF Y$= "R" THEN 90
74 GOTO 80
76 IF C$<>"R" THEN 90
77 IF Y$= "P" THEN 90
80 REM-------------- COMPUTER WINS
82 PRINT "                              COMPUTER WINS"
84 GOTO 30
```

```
90 REM
91 REM-------------- YOU WIN
92 REM
95 PRINT " YOU WIN"
99 GOTO 30
```

13A-1

```
10 REM HAPPY
20 PRINT "ARE YOU HAPPY? <Y/N>"
30 INPUT A$
40 IF A$="N" THEN 20
50 IF A$<>"Y" THEN 20
60 PRINT "GOOD"
```

13B-1

```
10 REM BOYS AND GIRLS
15 CALL CLEAR
20 PRINT
25 PRINT "ARE YOU A BOY OR A GIRL?"
26 PRINT "ANSWER 'BOY' OR 'GIRL'"
30 INPUT A$
32 PRINT
35 IF A$<>"BOY" THEN 40
36 PRINT "SNIPS AND SNAILS"
37 GOTO 60
40 IF A$<>"GIRL" THEN 25
41 PRINT "SUGAR AND SPICE"
60 REM ALL DONE
```

13C-1

```
2 REM PIZZA
3 REM BY CHRIS CLARK, JR. AGE 14 GOING ON (YOU FIGURE IT
OUT)
4 CALL CLEAR"
5 PRINT "HALLO, AY AM MARIO, YOUR PIZZA MAN."
6 PRINT
7 PRINT "JUST TELL ME ZE GORY DETAILS AND I'LL DO ZE REST"
9 PRINT
10 PRINT"WHAT SIZE SHOULD ZIS PIZZA BE? (S/M/L)"
20 INPUT S$
21 PRINT
30 IF S$<>"S" THEN 33
31 PRINT"ON A DIET? HO HO!"
```

```
33 IF S$<>"M" THEN 38
34 PRINT "GOOD CHOICE-NOT TOO BIG, BUT FILLING!"
38 IF S$<>"L" THEN 10
39 PRINT "YOU MUST HAVE A BIG BUNCH AT HOME!"
40 PRINT
41 PRINT "NOW, YOU WANT DOUBLE CHEES ON ZIS (Y/N)?"
42 INPUT CH$
45 REM ETC.
50 REM MUSHROOMS, ETC.
60 REM ANCHOVIES, ETC.
80 REM PEPPERS, ETC.
90 REM MEAT, ETC.
150 PRINT "HOKAY, HERE IS YOUR PIZZA!"
154 IF S$<>"S" THEN 156
155 PRINT "WAN SMALL PIZZA WITH ";
156 REM ETC.
160 IF BASE$<>"P" THEN 165
161 PRINT "PEPPERONI"
165 REM ETC., ETC.
238 PRINT
240 FOR J=1 TO 1000
242 NEXT J
```

13C-2

```
10 REM === COLOR GUESSING GAME ===
20 CALL CLEAR
23 PRINT
24 PRINT
25 PRINT "PLAYER 2 TURN YOUR BACK"
27 PRINT
30 PRINT "PLAYER 1 ENTER A COLOR"
35 INPUT C$
40 CALL CLEAR
42 PRINT
43 PRINT
50 PRINT "PLAYER 2 TURN AROUND AND GUESS"
52 PRINT
54 PRINT
55 INPUT G$
60 IF G$=C$ THEN 80
61 PRINT "WRONG!"
67 PRINT
70 GOTO 55
80 PRINT "RIGHT"
```

15A-2

```
10 REM !!! VACATION !!!
13 CALL CLEAR
14 PRINT
15 PRINT
16 PRINT
20 REM HEADING
21 PRINT "VACATION CHOOSING PROGRAM "
22 PRINT
23 PRINT "PICKS YOUR VACATION BY THE"
24 PRINT "AMOUNT YOU WANT TO SPEND"
25 PRINT
30 REM INSTRUCTIONS
31 PRINT "ENTER THE AMOUNT IN DOLLARS THAT "
32 PRINT "YOU CAN SPEND"
33 PRINT
35 REM GET DOLLAR AMOUNT
37 INPUT D
38 PRINT
40 M$="FLIP PENNIES WITH YOUR KID BROTHER"
41 P$="SPEND THE AFTERNOON IN BEAUTIFUL HOG WALLOW, MICH."
42 Q$="ENTER A PICKLE EATING CONTEST IN SCRATCHY
BACK,TENN."
47 REM ETC.
58 Z$="BUY A COSY YACHT AND CRUISE THE CARIBBEAN SEA"
70 IF D>0.5 THEN 75
71 PRINT M$
72 GOTO 900
75 IF D>1 THEN 80
76 PRINT P$
77 GOTO 900
80 IF D>5 THEN 85
81 PRINT Q$
82 GOTO 900
85 REM ETC.
300 IF D<1000000 THEN PRINT Z$:GOTO 90
301 PRINT "TREAT YOUR WHOLE SCHOOL TO A 'ROUND THE WORLD
TRIP!"
900 REM ENDING OF PROGRAM
```

15A-3

```
10 REM CRAZY
15 CALL CLEAR
20 PRINT "WHAT IS YOUR NAME?"
21 PRINT
```

```
22 INPUT N$
30 CALL CLEAR
40 PRINT
41 PRINT N$
45 PRINT
49 RANDOMIZE
50 Z=INT(RND*3)+1
60 ON Z GOTO 70,80,90
70 PRINT "HAS ONE BRICK SHORT OF A FULL LOAD"
71 END
80 PRINT "HAS BATS IN THE ATTIC"
81 END
90 PRINT "HASN'T GOT BOTH OARS IN THE WATER"
91 END
```

16-3

```
10 REM I GOT YOUR NUMBER!
20 CALL CLEAR
25 PRINT
26 PRINT
27 PRINT
30 PRINT "GIVE ME A NUMBER BETWEEN ZERO AND TEN:"
35 PRINT
36 PRINT
37 PRINT
40 INPUT N
45 PRINT
46 PRINT
50 IF N>0 THEN 60
51 PRINT "I GOT PLENTY OF NOTHING!"
52 GOTO 25
60 IF N>1 THEN 70
61 PRINT "I'M NUMBER ONE!"
62 GOTO 25
70 IF N=2 THEN 80
71 PRINT "TWO IS COMPANY"
72 GOTO 25
80 REM ETC.
160 IF N>10 THEN GOTO 999
161 GOTO 25
999 PRINT "THAT'S ALL, FOLKS"
```

16-4

```
10 REM CLOCK
15 CALL CLEAR
20 PRINT "TIME? <H, M, S>"
```

```
25 INPUT H, M, S
30 FOR T=1 TO 225
31 NEXT T
33 S=S+1
35 PRINT H;":";M;":";S
50 IF S<60 THEN 60
55 S=0
56 M=M+1
60 GOTO 30
70 REM DO SAME FOR HOURS
```

17B-3

```
2 GOTO 1000:REM SIMBAD'S MAGIC CARPET
198 REM
199 REM MAIN LOOP
200 REM
210 FOR I=1 TO 11
211 FOR J=1 TO 11
213 K=I+J-1
214 C=C+8
218 IF C<159 THEN 220
219 C=42
220 CALL HCHAR(I,K+5,C)
221 CALL HCHAR(K,I+5,C)
222 CALL HCHAR(23-I,K+5,C)
224 CALL HCHAR(K,28-I,C)
225 CALL HCHAR(I,28-K,C)
226 CALL HCHAR(23-I,28-K,C)
227 CALL HCHAR(23-K,28-I,C)
228 CALL VCHAR(23-I,28-K,C)
229 CALL VCHAR(23-K,28-I,C)
290 NEXT J
291 NEXT I
999 GOTO 999
1000 REM
1001 REM SINBAD'S MAGIC CARPET
1002 REM
2000 CALL CLEAR
2001 CALL SCREEN(16)
3000 FOR I=2 TO 16
3004 CALL COLOR(I,I,I)
3010 NEXT I
3200 C=42
3990 CALL CLEAR
3999 GOTO 200
```

19-1

```
10 REM RELATIVES
12 CALL CLEAR
20 PRINT "RELATION?"
21 PRINT
22 INPUT W$
23 PRINT
24 FLAG=0
29 RESTORE
30 READ R$
32 READ N$
34 IF R$="END" THEN 300
36 IF R$=W$ THEN 200
39 GOTO 30
90 DATA FATHER, WILLIAM
91 DATA MOTHER, ANNE
92 DATA SISTER, JOAN
93 DATA SISTER, SUSAN
94 DATA GRANDFATHER, JOHN
95 DATA GRANDMOTHER, ADA
96 DATA GRANDMOTHER, VIVIAN
97 DATA UNCLE, FRED
98 DATA UNCLE, GEORGE
99 DATA AUNT, MARY
100 DATA COUSIN, ROGER
110 DATA END, END
200 REM
201 REM PRINT IT
202 REM
210 PRINT R$;" ";N$
220 FLAG=1
299 GOTO 30
300 REM
301 REM NO RELATION
302 REM
310 IF FLAG=1 THEN 320
315 PRINT "YOU DO NOT HAVE A ";W$
320 FOR T=1 TO 200
321 NEXT T
399 GOTO 20
```

20-3

```
10 REM SONG
15 CALL CLEAR
20 FOR I=1 TO 10
22 READ P,D
```

```
24 CALL SOUND(D,P,10)
40 NEXT I
100 DATA 175,300,175,300,175,200,196,100
101 DATA 220,300,220,200,196,100,220,200,247,100,262,600
```

22-1

```
10 REM ALPHABETICAL
12 CALL CLEAR
20 PRINT "THIS PROGRAM ARRANGES"
21 PRINT "THE LETTERS OF A WORD"
22 PRINT "IN ALPHABETICAL ORDER."
25 PRINT
30 PRINT "GIVE ME A WORD"
31 PRINT
32 INPUT W$
33 PRINT
35 L=LEN(W$)
39 K=1
40 FOR I=65 TO 65+26
41 REM TEST LETTERS IN ALPHABET
42 REM TO SEE IF IN WORD
45 FOR J=1 TO L
50 G=ASC(SEG$(W$,J,1))
55 IF G<>I THEN 60
56 H$=H$ & CHR$(G)
57 K=K+1
60 NEXT J
61 NEXT I
70 PRINT "HERE IT IS IN ALPHABETICAL ORDER:"
75 PRINT
80 PRINT " ";H$
```

22-2

```
10 REM %#! DOUBLE DUTCH !%#
12 CALL CLEAR
25 PRINT"GIVE ME A SENTENCE"
26 PRINT
27 INPUT S$
28 PRINT
30 L=LEN(S$)
50 FOR I=1 TO L
51 L$=SEG$(S$,I,1)
52 IF L$="A" THEN 72
```

```
53 IF L$="E" THEN 72
54 IF L$="I" THEN 72
55 IF L$="O" THEN 72
56 IF L$="U" THEN 72
69 SS$=SS$ & L$
72 NEXT I
76 PRINT "HERE IT IS IN DOUBLE DUTCH"
78 PRINT
80 PRINT SS$
```

22-3

```
10 REM "ON ... GOTO" SAMPLE
20 REM MAKE A MENU
22 CALL CLEAR
25 PRINT "MAKE YOUR CHOICE:"
27 PRINT
28 PRINT " <A> TAKE A NAP "
29 PRINT
30 PRINT " <B> EAT AN APPLE "
31 PRINT
32 PRINT " <C> CALL A FRIEND "
40 PRINT
42 CALL KEY(0,X,S)
44 IF X=-1 THEN 42
46 PRINT
48 X=X-64
50 ON X GOTO 60,70,80
52 GOTO 22
60 PRINT "YOUR BED IS NOT MADE!"
61 END
70 PRINT "YOUR SISTER ATE THE LAST ONE!"
71 END
80 PRINT "YOUR FATHER IS ON THE PHONE!"
81 END
```

23-1

```
10 REM MENU MAKER
12 CALL CLEAR
20 PRINT "WHICH COLOR DO YOU LIKE?"
21 PRINT
22 PRINT " <Y> YELLOW"
23 PRINT " <R> RED"
25 PRINT " <B> BLUE"
26 PRINT
```

```
30 CALL KEY(0,C,S)
31 IF C=-1 THEN 30
32 C$=CHR$(C)
35 IF C$<>"Y" THEN 40
36 C=12
37 GOTO 80
40 IF C$<>"R" THEN 45
41 C=7
42 GOTO 80
45 C=5
80 CALL SCREEN(C)
90 FOR T=1 TO 500
91 NEXT T
```

23-2

```
10 REM SILLY SENTENCES
12 CALL CLEAR
13 PRINT"SILLY SENTENCES"
14 PRINT
15 PRINT"WANT INSTRUCTIONS <Y/N>"
16 PRINT
18 GOSUB 200
20 IF Y$="Y" THEN 100
21 PRINT"THE SUBJECT: (END WITH A PERIOD)"
22 PRINT
23 GOSUB 300
33 PRINT "THE VERB: (END WITH A PERIOD"
34 PRINT
40 GOSUB 300
50 PRINT"THE OBJECT: (END WITH A PERIOD)"
51 PRINT
52 GOSUB 300
85 PRINT S$
99 END
100 CALL CLEAR
110 PRINT"THREE PLAYERS ENTER PARTS OF A SENTENCE":?
115 PRINT"NO PLAYER CAN SEE WHAT THE OTHERS ENTER":?
120 PRINT"THE FIRST ENTERS THE SUBJECT"
121 PRINT" (THE PERSON DOING SOMETHING)":?
125 PRINT"THE SECOND ENTERS THE VERB"
126 PRINT" (THE ACTION WORD)":?
130 PRINT"THE THIRD ENTERS THE OBJECT"
131 PRINT" (THE PERSON OR THING TO WHOM"
132 PRINT" THE ACTION IS DONE)":?
150 FOR T=1 TO 2000
151 NEXT T
```

```
199 GOTO 21
200 REM LOOK AT KEYBOARD
210 CALL KEY(0,Y,S)
220 IF Y=-1 THEN 210
230 Y$=CHR$(Y)
240 CALL KEY(0,Y,S)
250 IF S<>0 THEN 240
299 RETURN
300 REM GET A WORD
310 GOSUB 200
320 IF Y$="," THEN 390
330 S$=S$ & Y$
340 GOTO 310
390 S$=S$ & " "
399 RETURN
```

24A-1

```
10 REM------------- GOSUB AND RETURN
12 CALL CLEAR
20 REM------------- THE FIRST ONE
21 GOSUB 200
30 REM------------- NEXT ONE
31 GOSUB 300
40 REM------------- THE LAST ONE
41 GOSUB 400
50 REM------------- AGAIN
51 GOSUB 200
99 END
200 REM
201 REM------------ SUBROUTINE 1
202 REM
210 PRINT "LOOK OUT!"
215 PRINT
250 GOSUB 900
299 RETURN
300 REM
301 REM------------ SUBROUTINE 2
302 REM
350 PRINT "RED SMOKE"
355 PRINT
360 GOSUB 900
399 RETURN
400 REM
401 REM------------ LAST ONE
402 REM
450 PRINT "IS POURING FROM YOUR COMPUTER!"
```

```
455 PRINT
460 GOSUB 900
499 RETURN
900 REM
901 REM------------ TIMER
902 REM
930 CALL SOUND(300,800,10)
950 FOR T=1 TO 400
951 NEXT T
999 RETURN
```

26-2

```
10 REM CIPHER MAKER
12 CALL CLEAR
20 PRINT "CODE MAKING PROGRAM "
21 PRINT
25 PRINT "ENTER A SENTENCE FOR CODING:"
26 PRINT
30 INPUT S$
35 L=LEN(S$)
36 S$=S$ & " "
40 FOR I=1 TO L STEP 2
45 P$=SEG$(S$,I,2)
50 Q$=SEG$(P$,2,1) & SEG$(P$,1,1)
55 L$=L$ & Q$
60 NEXT I
64 PRINT
65 PRINT "HERE IS THE CODED SENTENCE:"
66 PRINT
70 PRINT " ";L$
```

26-3

```
10 REM QUESTION ANSWERER
12 CALL CLEAR
20 PRINT "ENTER A QUESTION"
22 PRINT
25 INPUT Q$
27 L=LEN(Q$)
28 PRINT
30 REM TAKE OFF THE QUESTION MARK
32 Q$=SEG$(Q$,1,L-1) & ","
36 REM LOOK FOR THE END OF THE FIRST WORD
40 FOR I=1 TO L
41 C$=SEG$(Q$,I,1)
43 IF C$<>" " THEN 46
```

```
44 S1=I
45 I=L
46 NEXT I
48 REM LOOK FOR THE END OF THE SECOND WORD
50 FOR I=S1 + 1 TO L
52 C$=SEG$(Q$,I,1)
53 IF C$<>" " THEN 56
54 S2=I
55 I=L
56 NEXT I
58 REM TURN THE WORDS AROUND
60 S$=SEG$(Q$,S1+1,S2-S1)
62 V$=SEG$(Q$,1,S1)
65 PRINT S$;V$;SEG$(Q$,S2+1,L-S2)
```

26-4

```
10 REM PIG LATIN
15 CALL CLEAR
20 PRINT "PIG LATIN PROGRAM"
25 PRINT
30 PRINT "GIVE ME A WORD"
31 PRINT
33 INPUT W$
34 L=LEN(W$)
35 PRINT
40 REM FIND THE FIRST VOWEL
41 FOR I=1 TO L
42 V$=SEG$(W$,I,1)
43 IF V$="A" THEN 50
44 IF V$="E" THEN 50
45 IF V$="I" THEN 50
46 IF V$="O" THEN 50
47 IF V$="U" THEN 50
49 NEXT I
50 IF I<>1 THEN 60
52 L$=W$ & "LAY"
55 GOTO 80
60 REM FOUND IT
68 L$=SEG$(W$,I,L-I+1)
70 L$=L$ & SEG$(W$,1,I-1)
72 L$=L$ & "AY"
80 PRINT " ";L$
90 FOR T=1 TO 1000
91 NEXT T
99 GOTO 15
```

27-1

```
10 REM BACKWARD ADDED TO FORWARD
15 CALL CLEAR
20 PRINT "GIVE ME A NUMBER"
21 INPUT N
22 N$=STR$(N)
35 L=LEN(N$)
40 FOR I=1 TO L
41 B=L-I+1
45 B$=B$ & SEG$(N$,B,1)
50 NEXT I
55 B=VAL(B$)
57 PRINT
60 PRINT " ";N
61 PRINT " +";B
62 L$="-------------"
65 PRINT " "; SEG$(L$,1,L+2)
70 A=N+B
72 A$=STR$(A)
75 IF LEN(A$)<>L THEN 80
76 PRINT " ";A
77 END
80 PRINT A
```

27-2

```
10 REM MARCHING NUMBERS
15 CALL CLEAR
20 PRINT "GIVE ME A NUMBER "
21 PRINT
22 INPUT N
23 CALL CLEAR
25 N$=STR$(N)
26 L=LEN(N$)
40 FOR I=1 TO 32-L
43 B$=SEG$(N$,2,L)
44 B$=B$ & SEG$(N$,1,1)
45 N$=B$
50 C$=SEG$(N$,L,1)
52 C=ASC(C$)
56 IF I<L+1 THEN 60
58 CALL HCHAR(12,I-L,32)
60 CALL HCHAR(12,I,C)
65 FOR T=1 TO 100
66 NEXT T
70 NEXT I
```

30-1

```
10 REM ARRAYS
12 DIM D(12)
15 CALL CLEAR
16 PRINT
17 PRINT
18 PRINT
20 FOR I=1 TO 12
22 READ D
24 D(I)=D
29 NEXT I
30 PRINT "MONTH NUMBER? <1-12>"
31 PRINT
32 INPUT M
33 PRINT
35 PRINT "MONTH NUMBER ";M;" HAS ";D(M);" DAYS."
90 DATA 31,28,31,30,31,30,31,31,30,31,30,31
```

31B-2

```
10 REM =========== "AIN'T GOT NO ..." ==========
15 CALL CLEAR
16 PRINT
17 PRINT
18 PRINT
19 REM --------------------- GET A SENTENCE
20 PRINT "ENTER A SENTENCE:"
22 PRINT
23 INPUT S$
24 S$=S$ & " "
25 PRINT
26 L=LEN(S$)
29 REM --------------------- REMOVE PUNCTUATION
30 FOR I=1 TO L
31 L$=SEG$(S$,I,1)
32 C=ASC(L$)
33 IF C=39 THEN 38
34 IF C=32 THEN 38
35 IF C<65 THEN 39
36 IF C>89 THEN 39
38 C$=C$ & L$
39 NEXT I
40 REM NN IS NUMBER OF NEGATIVE WORDS
41 REM S2 IS START LETTER OF WORD
42 NN=0
43 S2=1
44 REM --------------------- TEST WORDS IN SENTENCE
```

```
45 FOR I=1 TO L
50 L$=SEG$(S$,I,1)
54 REM -------------------- IS IT A SPACE?
55 IF L$<>" " THEN 60
57 S1=S2+2
58 S2=I-1
59 GOSUB 200
60 NEXT I
65 REM -------------------- PRINT RESULT
66 PRINT
70 IF NN>0 THEN 75
71 PRINT "THIS SENTENCE IS POSITIVE."
72 GOTO 99
75 IF NN>1 THEN 80
76 PRINT "THIS SENTENCE IS NEGATIVE."
77 GOTO 99
80 IF NN>2 THEN 85
81 PRINT "THIS SENTENCE HAS A DOUBLE NEGATIVE."
82 GOTO 99
85 PRINT "THIS SENTENCE IS HARD TO UNDERSTAND!"
99 END
100 REM
101 REM -------------------- SOME TEST SENTENCES
102 REM
111 REM I DON'T EAT JUNK FOOD.
112 REM I NEVER EAT NO JUNK FOOD!
113 REM I DON'T NEVER EAT NO JUNK FOOD!
200 REM
201 REM -------------------- IS THE WORD NEGATIVE?
202 REM
205 RESTORE
210 W$=SEG$(S$,S1,S2-S1+1)
211 PRINT W$
215 READ N$
220 IF N$="END" THEN 299
225 IF N$<>W$ THEN 215
226 NN=NN+1
299 RETURN
900 REM
901 REM -------------------- NEGATIVE WORDS
902 REM
910 DATA NO,NOT,NEVER,NONE,NOTHING
911 DATA DON'T,DOESN'T,AREN'T,AIN'T
912 DATA ISN'T,DIDN'T
913 DATA HAVEN'T,HASN'T,HADN'T
914 DATA CAN'T,COULDN'T,SHOULDN'T
915 DATA WOULDN'T,WON'T
920 DATA END
```

32-2

```
2 REM **** CODE--DECODE ****
3 GOTO 1000
100 REM
101 REM                        MAIN LOOP
102 REM
109 REM ------------------ GET PASSWORD
110 GOSUB 400
115 PRINT
116 PRINT "CODE OR DECODE? <C/D>"
117 CALL KEY(0,Y,S)
118 IF Y=-1 THEN 116
119 Y$=CHR$(Y)
120 IF Y$="C" THEN 500:REM CODE MESSAGE
125 IF Y$="D" THEN 600:REM DECODE MESSAGE
130 GOTO 115
199 END
400 REM
401 REM ------------ FORM CODE ALPHABET
402 REM
405 PRINT "INPUT PASSWORD "
406 PRINT
407 INPUT PW$
408 REM ------------ REMOVE REPEATED LETTERS
409 F$=SEG$(PW$,I,1)
410 FOR I=2 TO LEN(PW$)
411 L1$=PW$(I,I)
412 FOR J=1 TO LEN(F$)
415 L2$=SEG$(F$,J,1)
420 IF L1$=L2$ THEN 430
421 NEXT J
422 F$ = F$&L1$
430 NEXT I
431 PW$=F$
432 PRINT
433 PRINT " THE SHORTENED PASSWORD IS "
434 PRINT
435 PRINT " ";PW$
436 REM REMOVE PASSWORD LETTERS FROM THE ALPHABET
440 FOR J=1 TO 26
442 L1$=SEG$(A$,J,1)
444 FLAG=0
446 FOR I=1 TO LEN(PW$)
447 IF SEG$(PW$,I,1)<>L1$ THEN 450
448 FLAG=1
449 I=LEN(PW$)
450 NEXT I
452 IF FLAG=1 THEN 455
```

```
454 D$=D$ & L1$
455 NEXT J
470 A$=PW$ & D$
772 PRINT
475 PRINT "ALPHABETS"
476 PRINT
480 PRINT "CIPHER"
481 PRINT
482 PRINT A$
483 PRINT
485 PRINT B$
486 PRINT
487 PRINT "NORMAL"
499 RETURN
500 REM
501 REM ----------------- FORM A CODED MESSAGE
502 REM
504 PRINT
505 PRINT "INPUT MESSAGE, END WITH '*'"
506 PRINT
510 CALL KEY(0,Y,S)
511 IF Y=-1 THEN 510
512 Y$=CHR$(Y)
515 IF Y$="*" THEN 590
520 IF Y<65 THEN 540
521 IF Y>89 THEN 540
525 Y$=SEG$(A$,Y-64,1)
540 P$=P$ & Y$
545 GOTO 510
590 PRINT
591 PRINT P$
591 PRINT
599 END
600 REM
601 REM ------------------ DECODE A MESSAGE
602 REM
609 PRINT
610 PRINT "TYPE IN THE CODED MESSAGE"
611 PRINT
612 PRINT "END WITH A '*' SIGN"
613 PRINT
615 CALL KEY(0,Y,S)
616 IF Y=-1 THEN 615
617 Y$=CHR$(Y)
618 IF Y$="*" THEN 690
620 FOR I=1 TO 26
625 IF Y$=<>SEG$(A$I,1) THEN 640
630 Y$=SEG$(B$,I,1)
```

```
631 I=26
640 NEXT I
645 PRINT Y$
650 GOTO 615
690 END
1000 REM
1001 REM      **** CODE--DECODE ****
1002 REM
2003 CALL CLEAR
2004 PRINT
2010 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2015 B$=A$
2999 GOTO 100
```

EXTRA

```
10 REM +++ WALL PAPER +++
11 CALL CLEAR
12 RANDOMIZE
15 READ X$
20 Q$=""
22 FOR J=1 TO 16
25 Z=INT(RND*16)+1
26 B$=SEG$(X$,Z,1)
30 Q$=Q$ & B$
35 NEXT J
40 CALL CHAR(32,Q$)
42 X=INT(RND*16)+1
44 Y=INT(RND*16)+1
46 IF X=Y THEN 44
47 FOR T=1 TO 500
48 NEXT T
50 CALL COLOR(1,X,Y)
99 GOTO 20
100 DATA 0123456789ABCDEF
```

GLOSSARY

argument
The variable, number or string which appears in the parentheses of a function. Like:

`INT(N)`	has	`N`	as an argument
`LEN(W$)`	has	`W$`	as an argument

array
A set of variables which have the same name. The members of the array are numbered. The numbers appear in parentheses after the variable name. See dimension, subscript. Examples:

`A(0)`	is the first member of the array A
`B$(7)`	is the eighth member of the array B$
`CD(3,I)`	is a member of the array CD

arrow keys
Four keys on the computer which have arrows on them. They move the input cursor to the left and right. The up and down arrow keys are used to go to the previous or the next lower lines when in the line edit mode.

ASCII
Stands for American Standard Code For Information Interchange. Each character has an ASCII number.

assertion
The name of a phrase which can be TRUE or FALSE. The "phrase A" in an IF statement is an assertion. An assertion is also a numeric expression with value 0 or −1. See expression, TRUE, FALSE, logic, IF, phrase A. Example:

the assertion	`"A"<>"B"`	is	`TRUE`
the assertion	`3 = 4`	is	`FALSE`

background
The part of the screen which is blank, not having characters on it.

BASIC
Beginners's All-purpose Symbolic Instruction Code. A computer language originated by John Kemeny and Thomas Kurtz at Dartmouth College in the early '60s.

bell
The early teletype machines had a bell (like the bell on a typewriter). The TI 99/4A makes a low tone sound instead.

bells and whistles
A phrase going back to the early days of hobby computing. It means the personal computer was hooked up to do some interesting or spectacular things, like flash lights or play music.

blank
The character which is a space.

boot

Means to start up the computer from scratch. An easy thing to do with modern computers which have start up programs stored permanently in ROM memory. It was an involved procedure in the early days. Now it usually means to read in the disk operating system programs (DOS) from a disk.

branch

A point in a program where there is a choice of which statement to execute next. An IF statement is a branch. So is an ON...GOTO statement. A branch is not the same as a jump where there is no choice. See jump, IF.

buffer

A storage area in memory for temporary storage of information being inputed or outputed from the computer.

call

Using a GOSUB calls the subroutine. Putting a function into a statement calls the function. Call means the computer does what commands are in the subroutine or does the calculation which the function is for.

carriage return

On a typewriter, you push the lever which moves the carriage carrying the paper so a new line can begin. In computing, it means the cursor is moved to the start of the line, but not down to the next line. See line feed, CRLF.

character

Letters, digits, punctuation marks and the space are characters. So are graphics characters you make with CALL CHAR( ).

checksum

In some I/O operations, the computer adds together all the character numbers. The resulting sum is the "checksum." If the data was transmitted correctly, the checksum calculated after the data is received will agree with that calculated before the data was sent. See I/O.

clear

Means erase. Used in "clear the screen" and "clear memory."

coldstart

When first turned on, the computer loads certain parts of memory. Some of these will be changed as the computer is used. See warmstart.

column

Things arranged vertically. See row.

command

In BASIC a command makes the computer do some action, such as erase the screen and memory by the NEW command. See statement, expression. Some commands need expressions to be complete. Example:

```
IF X+3 THEN 35
```

concatenation

Means sticking two strings together.

constant
A number or string which does not change as the program runs. It is stored right in the program line, not in a box with a name on the front. See line.

CRLF
Short for "carriage return followed by line feed." This is what is called just a "carriage return" on a typewriter. See carriage return, line feed.

cursor
A marker which shows where the next character on the screen or in a storage buffer will be placed. Cursor means "runner." The cursor runs along the screen as you type. There are two kinds of cursors in the TI:

INPUT cursor	a flashing square on the screen
PRINT cursor	invisible, "shows" where next character will be printed

data
BASIC has two kinds of data: numeric and string. Logical data (TRUE, FALSE) are types of numeric data.

debug
Means to run a program to find the errors and fix them. You fix the errors by editing the program. See edit.

deferred execution
Means run a stored program. See immediate execution

delay loop
A part of the program which just uses up time and does nothing else. Example:

```
10 FOR T=1 TO 2000
31 NEXT T
```

duration
A number in a CALL SOUND( ) statement which tells how long the sound will last.

edit
There are two kinds: editing a line and editing a program. In either kind you are retyping parts of it to correct it.

enter
To put information into the computer by typing, then pushing the ENTER key. The information goes into the input buffer as it is typed. When ENTER is pushed, the computer uses the information.

erase
To destroy information in memory or write blanks to the screen. See clear.

error trap
Part of a program which checks for mistakes in information that the user has entered, or checks to see if computed results make sense.

execute
To run a program or to perform a single command or statement.

expression
A portion of a statement which has a single value, either a number or a string. See value. Examples:

```
7*X+1
"DOPE "<> N$
A$ & "HAT"
```

FALSE
The number 0. See logic, assertion.

fork in the road
Describes a branch point in the program. See branch.

function
BASIC has a number of functions built in. Each function has a name followed by parentheses. In the parentheses are one or more arguments. The function has a single value (numerical or string) determined by its arguments. See value, argument. The functions treated in this book are:

```
ASC, CHR$, INT, LEN, POS, RND, SEG$, STR$, VAL
```

garbage
A random mess of characters in memory. Usually due to human or machine error.

graphics
Means picture drawing.

immediate execution
When a command which does not start with a number is entered from the command mode, it is executed as soon as the ENTER key is pressed.

index
An array name is followed by one or more numbers or numerical variables in parentheses. Each number is an index. Another word for index is "subscript."

```
Q(7,I)          7 and I are indices
```

integers
The whole numbers, positive, negative and zero.

I/O
Input/Output. Input from keyboard, tape recorder, etc. Output to screen, printer, tape recorder, etc.

joystick
A device used in games. It is like the control stick used in early airplanes. It can detect 8 different directions, as well as "centered." Texas Instruments calls its joysticks "Wired Remote Controllers."

jump
The GOTO command makes the computer jump to another line in the program, rather than execute the next line.

line

Lines start with a number followed by a statement which may contain expressions, etc.

Parts of a line:

```
16 IF 7<=INT(Z) THEN 48
```

16	line number
IF 7<=INT(Z) THEN 48	statement
7<=INT(Z)	an assertion
7<=INT(Z)	an expression
INT(Z)	a function
Z	argument
7	constant
<=	operation
IF, INT, THEN	reserved words

line buffer

The storage space which receives the characters you type in. See buffer.

line editing

Retyping parts of a line to correct it. First you call up the line with EDIT nnn, where nnn is the line number. Then you move the cursor to the wrong part and type the correct characters. Store the corrected line in memory by pressing ENTER.

line feed

Moving the cursor straight down to the next line. The ASCII number 10 signals this command to the screen or printer. See carriage return and CRLF.

line numbers

The number at the beginning of a program line. The line number tells the computer where to store the line. Some lines don't have numbers (the ones which will be executed in the immediate execution mode).

listing

A list of all the lines in a program.

load

To transfer the information from a file on tape to the memory of the computer by using the OLD command.

logic

The part of a program which compares numbers or strings. The relations =, <>, <, >, <=, and >= are used. See assertion, IF, phrase A.

loop

A part of the program which is done over and over again. There are many kinds of loops: FOR...NEXT loops, "home made" loops which use IF... commands with a loop variable and DO WHILE... and DO UNTIL... loops.

loop variable
Is the number which changes as the loop is repeated. For example:

```
40 FOR I=1 TO 5
50 NEXT I
```

I is the loop variable

loudness
A number in CALL SOUND which tells how loud the sound should be.

memory
The part of the computer where information is stored. Memory is made of semiconductor chips, but we think of it as "boxes" with labels on the front and information inside.

menu
A list of choices shown on the screen. Each choice has a letter or number beside it. The program user presses a key to pick which choice is wanted.

message
A PRINT statement which tells what is expected in an INPUT statement. Example:

```
60 PRINT "AGE?"
61 INPUT A
```

monitor
Has two meanings. We use it to mean a box with a TV type screen which is connected to the computer. It displays text and graphics but cannot receive television programs. In machine language programming, a monitor is a control program.

nesting
When one thing is inside of another. In a program we nest loops. Inside a statement, we can nest expressions or functions.

```
L=INT(LEN(P$)+3.4)          nested functions
X=5*(6+(7*(8+K)))           nested parentheses
```

number
Is one type of information in BASIC. The numbers are generally decimal numbers. See integer, strings.

operation
In arithmetic: addition, subtraction, multiplication and division, with symbols +, −, *, and /.The only operation for strings is concatenation.

phrase A
Is a phrase in this book which stands for an assertion in an IF statement. See assertion, IF. Example:

```
IF A>4 THEN 500          A>4     is "phrase A"
```

pitch
The number in a CALL SOUND( ) statement which tells the musical pitch of the sound. Pitch is the same as "note" and can be high or low.

pixel
PIcture Element. The smallest dot which is placed on the screen in a graphics mode.

pointer
A number in memory which tells where in a list of DATA you are at the present moment.

program
The usual program is a list of numbered lines containing statements. The computer executes the statements (commands) in order when the RUN command is entered. The program is stored in a special part of memory; only one program can be stored at a time.

prompt
Is a little message you put on the screen with an INPUT to remind the user what kind of an answer you expect. Its name comes from the hint that actors in a play get from the prompter if they forget their lines.

pseudo-random
A number which is calculated in secret by the computer using the RND function. It is usually called a "random number." Pseudo-random emphasizes that the number really is not random (since it is calculated by a known method) but just is not predictable by the computer user.

punctuation
The characters like period, comma, /, ?, !, $, etc.

random
Numbers which cannot be predicted, like the numbers that show after the roll of dice, or the number of heads you get in tossing a coin 10 times.

remark
A comment you make in the program by putting it into a REM statement. Example:

```
REM THE GRAPHICS SETUP SUBROUTINE
```

reserved words
A list of words and abbreviations that BASIC recognizes as commands, statements, or functions. The reserved words cannot be used as variable names.

return a value
When a function is used (called), its spot in the expression is replaced with a value (a number or a string). This is called "returning a value."

RUN mode
The action of the computer when it is executing a program is called "operating in the RUN mode." You get into the run mode from the command mode by entering RUN. When the computer ends the program for any reason, it returns to the command mode.

row
Things arranged horizontally (across).

save
To save a program which is in memory on tape.

screen
The TV screen or a similar one in a monitor which is hooked up to the computer. See monitor.

scrolling
The usual way the computer writes to the full screen is to put the new line at the bottom of the screen and push all the old lines up. This is called "scrolling."

simple variable
A variable which is not an array variable.

stack
Is a data type used in machine language programming. The data are arranged in a column and the last one put on is the first one taken off.

starting stuff
Is the name given in this book to initialization material in a program. It includes REMs for describing the program, input of initial values of variables, set up of array dimensions, drawing screen graphics, and any other things which need to be done just once at the beginning of a program run.

statement
The smallest complete section of a program. It starts with a command. The command may have expressions in it.

store
To put information in memory or to save it on tape.

string
A type of data in BASIC. It consists of a row of characters. See number.

subroutine
A section of a program which starts with a line called from a GOSUB command and ends with a RETURN command. It may be called from more than one place in the program.

subscript
A number in the parentheses of an array. It tells which member of the array is being used. See index.

syntax
Means the way a statement in BASIC is spelled. A syntax error means the spelling of a command or variable name is wrong, the punctuation is wrong or the order of parts in the line is wrong.

timing loop
A loop which does nothing except use up a certain amount of time. See delay loop.

title
The name of a program or subroutine. Put it into a REM statement.

TRUE
Has the value −1. See logic, FALSE, assertion.

typing
Pressing keys on the computer. It is different from "entering." See enter.

value
The value of a variable is the number or string stored in the memory box belonging to the variable. See variable.

variable
A name given to a "box" in memory. The box holds a value. When the computer sees a variable name in an expression, it goes to the box, takes a copy of what is in the box back to the expression, puts it where the variable name was, and continues to evaluate the expression. See variable names.

variable name
A variable is either a string variable or a numerical variable. The name tells which. String variables have names ending in a "$" sign. Numerical variables do not.

wired remote controllers
See joystick.

RESERVED WORDS IN TI BASIC

ABS	APPEND	ASC	ATN	
BASE	BREAK	BYE		
CALL	CHR$	CLOSE	CON	
	CONTINUE	COS		
DATA	DEF	DELETE	DIM	DISPLAY
EDIT	ELSE	END	EOF	EXP
FIXED	FOR			
GO	GOSUB	GOTO		
IF	INPUT	INT	INTERNAL	
LEN	LET	LIST	LOG	
NEW	NEXT	NUM	NUMBER	
OLD	ON	OPEN	OPTION	OUTPUT
PERMANENT	POS	PRINT		
RANDOMIZE	READ	REC	RELATIVE	REM
	RES	RESEQUENCE	RESTORE	RETURN
	RND	RUN		
SAVE	SEG$	SEQUENTIAL	SGN	SIN
	SQR	STEP	STOP	STR$
	SUB			
TAB	TAN	THEN	TO	TRACE
UNBREAK	UNTRACE	UPDATE		
VAL	VARIABLE			

Variable names in TI BASIC can be up to 15 characters long.
You should not use reserved words as variable names.

INDEX OF COMMANDS AND FUNCTIONS EXPLAINED IN THIS BOOK

INDEX OF KEYS USED IN THIS BOOK

ERROR MESSAGES

ERRORS WHILE YOU ARE ENTERING A LINE

* BAD LINE NUMBER

You used a line number which was zero or larger than 32767.

* BAD NAME

You used a name which was more than 15 characters long
or you used a bad character in the name:

right	ABC
wrong	AB%

* CAN'T CONTINUE

You tried to CONTINUE a program but:
The program had not been stopped with a FCTN CLEAR key.

* CAN'T DO THAT

You tried to use one of these statements as a command:
DATA, FOR, GOTO, GOSUB, IF, INPUT, NEXT, ON, RETURN

or you tried to use one of these commands as a statement:
CONTINUE, EDIT, LIST, NEW, OLD, RUN, SAVE

or you entered LIST, RUN, or SAVE when there was no program in the memory.

* INCORRECT STATEMENT

You used incorrect punctuation in a statement. Maybe you forgot a :, ', =, +, ;, etc.

* LINE TOO LONG

Lines can be only 4 lines long on the screen.

* MEMORY FULL

Your program is too big.

ERRORS JUST AFTER YOU ENTER 'RUN'

The computer makes some checks of your program before you run it.

* BAD VALUE

You made an array with a dimension larger than 32767.

* FOR-NEXT ERROR

Your program does not have the same number of NEXT statements as FOR statements.

* INCORRECT STATEMENT

A DIM statements is:

missing its dimension number
or has more than 3 dimension numbers
or you forgot a comma or a) in the DIM
or you have a bad name in the DIM statement

* MEMORY FULL

You made an array which is too large
or there is no more room in memory for variables

* NAME CONFLICT

You named two different sized arrays with the same name
or you named a simple variable and an array the same
or you used DIM to make an array, then used it with a different number of subscripts

Wrong:
```
10 DIM A(3,3,3)
20 A(2,2)=5
```

ERRORS WHILE THE PROGRAM IS RUNNING

* BAD ARGUMENT

You used ASC or VAL with a string which was of zero length
or you put a string into VAL which was not a number

Wrong:
```
30 A=VAL("33.NN")
```

* BAD LINE NUMBER

You asked an ON, GOTO or GOSUB statement to goto a line which is not in the program.

* BAD NAME

You put a name after CALL which is not allowed.

The only allowed names are CHAR, CLEAR, COLOR, GCHAR, HCHAR, JOYST, KEY, SCREEN, SOUND, VCHAR

* BAD SUBSCRIPT

You have a bad number as an array subscript:
It is not an integer
or it is bigger than the size you put in the DIM command

Wrong:
```
10 DIM A(8)
20 A(10)=5
```

* BAD VALUE

The computer expected a number in the line, but didn't find it
or the number was negative when a positive one is needed
or the number had some letters mixed in
or the number was too big
or the number was zero when it must be positive
or the computer expected to see an = sign.

* INPUT ERROR

The computer came to an INPUT statement, then:
you input too long a line of characters
or you input more or fewer numbers than were asked for
or you typed letters when a number was asked for.

* I/O ERROR

You made an error with the commands SAVE, OLD, INPUT, PRINT, or RESTORE

* MEMORY FULL

There is too little memory to use with the CALL CHAR
or your GOSUB line goes to itself
or you nested your subroutines too deeply
or you have an expression which is too complicated.

* NUMBER TOO BIG

You made a number (positive or negative) bigger than 9.99 E 127.

* STRING-NUMBER MISMATCH

You used a string where a number was needed
or you used a number where a string was needed.

* CHECK PROGRAM IN MEMORY

This happens only when you commanded OLD and it did not put a program from tape into memory. It is telling you that the program in memory may be bad.

INDEX OF TOPICS

KIDS AND THE TI 99/4A

A KID'S BOOK FOR GROWN UPS?
YOU BET!

Learning Basic and the magic tricks of the TI computer can be great fun—so why learn it any other way?

The drawings help you understand and remember the material—so you'll be writing your own programs by Lesson five and forever after.

Example programs are easy to understand and useful to have. Your own programs can be easily made by combining parts from the programs in this book.

YOU WILL QUICKLY LEARN:

TO write computer music

TO create computer graphics and drawings in full color

TO make your computer talk and ask you questions

TO create games and quizzes

TO edit and debug programs fast!

TO organize your mind and think creatively

AND BEST OF ALL IT'S FUN

KIDS CAN DO IT . . . SO WHY NOT YOU?

DATAMOST™ INC

8943 Fullbright Ave., Chatsworth, CA 91311 (213) 709-1202

ISBN 0-88190-000-1